TRIBES® Script Programming

Jacob 'Vage' Gohlke

Version 3.25.05

TRIBES® Script Programming

Jacob Gohlke

I. Foreword
II. Introduction
- The Tools
- The Files
- The IDE
- The Language
- First Modification

III. Variables and Operators
- General Variable Information
- Numerical Variables
- Boolean Variables
- String Variables
- Object, Player, and Client Variables

IV. Decision and Looping Statements
- If-Else
- While Loop
- For Loop

V. Functions and Datablocks
- Functions
- Datablocks

VI. Arrays
- Arrays / Subscripted Variables

VII. Sorting and Searching
- Linear Search
- Numeric Sort
- Alphabetical / Lexicographic Sort

VIII. Bitwise
- Theory
- Bitwise Operators
- Uses for Bitwise

IX. Last Words
X. Appendices
- Function Reference

Chapter I
Foreword

1.1 Intro

TRIBES Script has been shrouded in mystery for a while now; the general public knows it exists, however only the elite few are extremely good at it. I've written this book on basically every subject that will help you become one of those elite few. I assume that you've got a basic working knowledge of a PC and the Windows operating system, you've at least played TRIBES before, and you have access to the internet. I use Windows XP, but any version beyond Windows 98 should work fine.

If you want to fiddle with server scripts or make the next 'Annihilation', this book is for you. This is the first edition, and there will be revisions if I find I made a mistake somewhere. If you find any, email me at: 'jbgohlke@yahoo.com'.

1.2 About the Author

Heh, well you probably want to know who'll be teaching you. I'm Jacob, or better known as Vage, and I've been coding and modding TRIBES since early 2004. I've created ModX and helped various coders with issues. I currently reside in Houston, TX.

Enjoy the book!

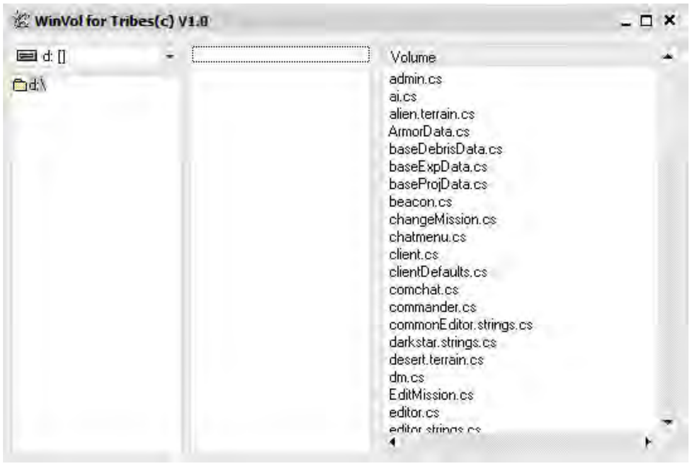
Chapter 2
Introduction

2.1 The Tools

To begin coding, you’re going to need several things, a few of which can be downloaded off of the internet:

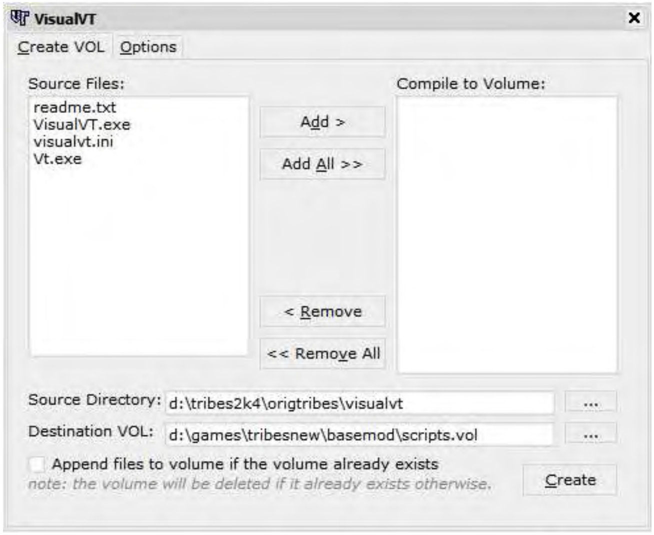
1) A VOL Extraction Program
2) A VOL Complication Program
3) A Text Editor, preferably with Syntax Coloring
4) Common Sense
5) Persistence and Patience

I can help you with the first three, but the others you have to work on by yourself. To get a VOL Extraction program, you can search for several different ones, such as Volumer or WinVOL. I personally use WinVOL for its simple interface. When you open WinVOL you’ll first be presented with a splash screen giving credit where it is rightfully deserved. Once at the next screen you’ll see something similar to the picture to the right. There are three different boxes here, from left to right: The Browser Box, The Directory Box, and the Volume Contents Box. The Browser Box allows you to browse trough your drives and folders to wherever you’d like to be. The Directory Box shows all VOL files within the current directory, and the Volume Contents well… displays the current volume’s contents. You can then select the files you’d like to extract and secondary-click on them. Voila, instant extraction! You’ll also need a VOL Complication program to make your own VOLs however. There are many out there, but I use VisualVT myself. On my


WinVol for Tribes(c) V1.0

d: []
d:\

Volume
admin.cs
ai.cs
alien.terrain.cs
ArmorData.cs
baseDebrisData.cs
baseExpData.cs
baseProjData.cs
beacon.cs
changeMission.cs
chatmenu.cs
client.cs
clientDefaults.cs
comchat.cs
commander.cs
commonEditor.strings.cs
darkstar.strings.cs
desert.terrain.cs
dm.cs
EditMission.cs
editor.cs
editor.strings.cs


VisualVT

Create VOL | Options

Source Files:
readme.txt
VisualVT.exe
visualvt.ini
Vt.exe

Compile to Volume:

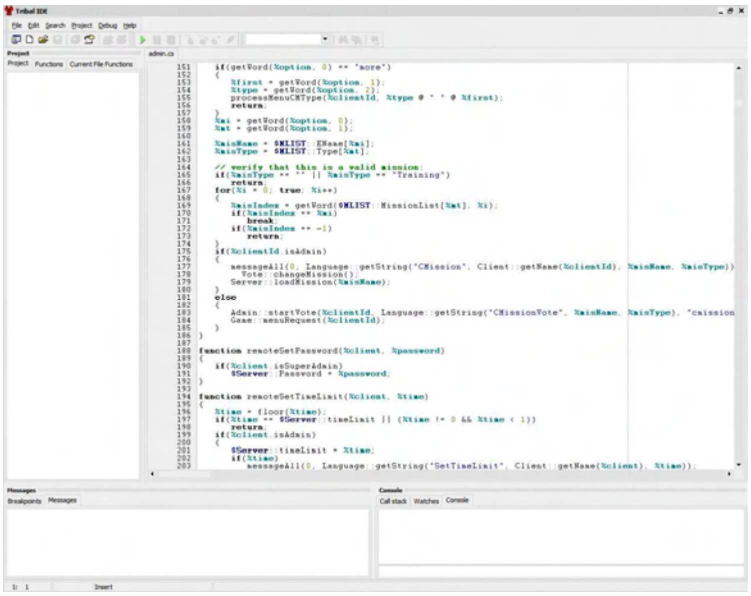
Add >
Add All >>
< Remove
<< Remove All

Source Directory: d:\tribes2k4\origtribes\visualvt ...
Destination VOL: d:\games\tribesnew\basemod\scripts.vol ...

Append files to volume if the volume already exists
note: the volume will be deleted if it already exists otherwise.

Create

bigger projects I use a batch file and the original VT program, but for learning purposes VisualVT should work just fine. If you care to have your own batch file, it's in the 'Code Snippets' section. This is a rather self-explanatory interface, so I won't go into detail on how to use it. Well, that's 2 out of 5! Let's go on. Next, you'll need a Text


Tribal IDE
File Edit Search Project Debug Help
Project
Project Functions Current File Functions
admin.cs
Messages
Breakpoints Messages
Console
Call stack Watches Console
Insert

```
    if(getWord(%option, 0) == "more")
    {
        %first = getWord(%option, 1);
        %type = getWord(%option, 2);
        processMenuCMType(%clientId, %type @ " " @ %first);
        return;
    }
    %mi = getWord(%option, 0);
    %mt = getWord(%option, 1);

    %misName = $MLIST::EName[%mi];
    %misType = $MLIST::Type[%mt];

    // verify that this is a valid mission:
    if(%misType == "" || %misType == "Training")
        return;
    for(%i = 0; true; %i++)
    {
        %misIndex = getWord($MLIST::MissionList[%mt], %i);
        if(%misIndex == %mi)
            break;
        if(%misIndex == -1)
            return;
    }
    if(%clientId.isAdmin)
    {
        messageAll(0, Language::getString("CMission", Client::getName(%clientId), %misName, %misType))
          Vote::changeMission();
        Server::loadMission(%misName);
    }
    else
    {
        Admin::startVote(%clientId, Language::getString("CMissionVote", %misName, %misType), "cmission
        Game::menuRequest(%clientId);
    }
}

function remoteSetPassword(%client, %password)
{
    if(%client.isSuperAdmin)
        $Server::Password = %password;
}

function remoteSetTimeLimit(%client, %time)
{
    %time = floor(%time);
    if(%time == $Server::timeLimit || (%time != 0 && %time < 1))
        return;
    if(%client.isAdmin)
    {
        $Server::timeLimit = %time;
        if(%time)
            messageAll(0, Language::getString("SetTimeLimit", Client::getName(%client), %time));
```

Editor of some sort. Now, Notepad will work just fine, but you'll find it will grind on your nerves after a while. I personally recommend Tribal IDE. It was originally make for Tribes 2, but it works just fine for TRIBES. The syntax coloring will give you a great advantage over Notepad because you'll be able to distinguish similar parts of code from one another. The above picture gives you a preview of what it looks like. Well, that's all I can help you with. The last two are out of my hands. I recommend a self-help book or some thing similar. Once you have *all* the tools, you are ready to begin modding.

2.2 The Files

Once you have all the tools you need, and are familiar with them, we can proceed. To start, CS files are Script files. They are the most common of file types and will hold all the code that you create. VOL files are basically like a container for the CS files: They hold everything together, instead of having lots of CS files floating around. To start modifying the 'base' mod, we're going to need to gain access to the CS files from the 'scripts.vol' file. The original script files should never be replaced, unless you have a recent backup. You need to open the VOL extractor you downloaded, and select the 'scripts.vol' file from your TRIBES directory. Don't see it? Well, first let's find your TRIBES directory. If you have a shortcut on your desktop or start menu to TRIBES, right-click on it and look at the target text box on the 'Shortcut' tab. That's your TRIBES directory. It will look something like this: 'X:\Program Files\Dynamix\TRIBES\'. The 'scripts.vol' is in the 'base' folder. Another alternative is to use Windows' Search feature. You need to now make a folder within the TRIBES directory with your Mod Name. For learning purposes, I'm going to use the folder name 'test'; I recommend you do the same. Now, go back to your extractor. Open up the 'scripts.vol' you copied with it, and extract *everything* from the file to your newly created folder. Once the VOL file is extracted, rename it to 'scripts.vol.bak'. This way, you can always restore from a messed-up mod or TRIBES. If you leave the file as-is, any changes will be overridden. Congratulations, you have a mod! Of course, it's nothing special, but we're going to change that, right? Next up is to create a shortcut to your mod. First, copy the existing one from your start menu or desktop onto the desktop (Or create a new one, depending on your circumstances). Rename this to something along the lines of 'TRIBES test mod' or whatever your mod name might be. Right-click on this shortcut and go to the 'Shortcut' tab. We are going to change the 'target' textbox. Because we don't want to run 'base' we want to run our mod, 'test', we need to change this from something like 'X:\Program Files\Dynamix\TRIBES\Tribes.exe' to 'X:\Program Files\Dynamix\TRIBES\Tribes.exe –mod test'; Where 'test' is, of course, your mod name. If you were to run TRIBES from this shortcut, your server type would be the mod name 'test' with 'base' at the end, and it would run your mod 'test'. If you didn't do it correctly, you might get a 'Game::EndFrame: Unknown command.' Error.

2.3 The IDE

The Integrated Development Environment or IDE for short is where you'll be spending most of your coding career. In the beginning of the chapter I told you to get a Text Editor. In TRIBES Script, the terms are nearly synonymous, In Tribal IDE, all the extras are Tribes 2 only, but it's still a very good editor. There are others, such as Crimson Editor or EditPad but you must create your own syntax grammars so they can properly highlight your code, which at a beginner level can be nearly impossible to do. So, I would recommend you use Tribal along with me. Play around with the program a while, get familiar with the interface and the various text editing tools. Once you're satisfied, move on.

2.4 The Language

The TRIBES Script language is an odd mix of PHP, Java, and C. If you've got experience in any of these languages, you should do just fine. For you others, there is a quite a learning curve but you should manage if you persist. A sample bit of code might look something like this:

```
     function Sample::Function(%msg)
     {
          echo(%msg);
     }
```

You can probably see the similarities if you have experience with another language and if not this is a good first look at a language. The syntax seems straight-forward enough, let's analyze it. The first thing you see it 'function'. This denotes that your starting a function block. The next is the identifier, 'Sample::Function'. This tells TRIBES what your function's name is. Then comes the argument list encased in parentheses, '%msg'. This is a list of variables that are used in the function. The rest is the body of the function encased in brackets. We'll analyze the language in more detail throughout the book. Syntax errors appear on the TRIBES console when your in-game. To gain access to the console, click '~' on your keyboard. You should see white text overlay itself over your game. You can type almost any script command in here.

2.4 Style and Readability

Style and readability is different for everyone, that's what makes us human and makes reading other's code hell. So, follow my style and everything should work out fine. If you ask others for help, say for instance on a forum or message board, be sure to have all your formatting issues worked out and solved. You'll get more response that way.

2.5 First Modification

Now we are going to make your first modification. You ready? Good. Go to your mod directory created earlier. Let's open up 'game.cs'. You'll notice a lot of new and foreboding things, don't worry I'll guide you though it. Now, scroll down to about line 200. You should see something similar to this:

```
$spawnBuyList[0] = LightArmor;
$spawnBuyList[1] = Blaster;
$spawnBuyList[2] = Chaingun;
$spawnBuyList[3] = Disclauncher;
$spawnBuyList[4] = RepairKit;
$spawnBuyList[5] = "";
```

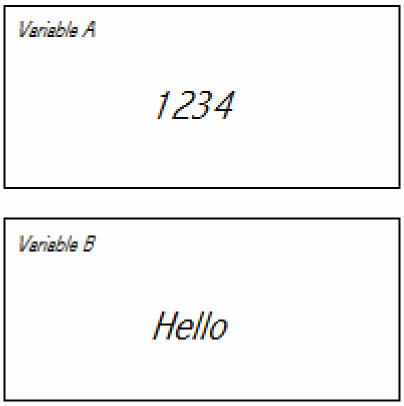
This is an array of items. You can probably guess what they are by their names. Arrays are explained in a later chapter, but just take them for what they are now. The '$spawnBuyList' global variable holds items that are bought when the player spawns or respawns. Let's change it, shall we? For this exercise we are going to modify the second variable from 'Blaster' to 'PlasmaGun'. Where did PlasmaGun come from you ask? Well, It's one of the items already programmed into 'base'. More details will follow in a later chapter. Well, our final variable should look like:

```
$spawnBuyList[1] = PlasmaGun;
```

Did you do it right? If not, don't fret, just fix it. Once you have that done. Save the modified file and run TRIBES from your mod shortcut. You should spawn with a Plasma Gun instead of a Blaster. Cool, eh? Well, it's not exactly a remote mine or Particle Beam Cannon, but it is a start.

Chapter 3
Variables and Operators

3.1 General Variable Information



Variables are a basic concept of Algebraic Mathematics, and Computer Science. You can think of variables as a container that can hold different things. This is illustrated in the image. As you can see, the variable has an identifier and a value, or constant. In TRIBES Script, a variable can be anything, and changes according to what it's storing. This is different from most languages where variables can only be certain types, such as numbers or a string of letters. Variables, in TRIBES Script, look like this:

```
     %variableName
```

To use a variable in TRIBES Script, you only have to type it and TRIBES will create it automatically. You cannot use a variable by itself, it must be used in a function or operation. Operations are performed with operators, which are explained for each variable type. To assign a value to a variable, you might write some code that looks like:

```
     %numberVariable = 1234;
     %stringVariable = "Hello";
```

This assigns 'numberVariable' to equal the number '1234' and 'stringVariable' to equal "Hello". As you see above, we are using the '=' character to assign a constant (ex. 1234) to a variable. This character, '=' is called the 'Assignment Operator', and can be used on any variable type with no ill effects. This is called 'Assignment'. You can also assign a variable to another variable. For example:

```
     %numberVariable = 1234;
     %stringVariable = %numberVariable;
```

stringVariable would be equal to 1234. Cool, eh? Also, global variables are declared like so:

```
     $globaVariable = 0;
```

globalVariable can be accessed anywhere. I should mention now that there are two types of variables, Global and Local. These are different because of something called 'scope'.

The scope of a variable depends on when TRIBES deletes it. A local variable is only available for use in the function it was created in, whereas a global variable, or global, can be used *anywhere*.

3.2 Numerical Variables

Numerical Variables are variables that store numbers, or a numeric value. To create a numerical variable just assign a number, whether it be '1234' or '45.87983' to a variable. Once you have a numerical variable, you can perform arithmetic with it. There are several operators you can use for this type, the most basic of which I'll list here, assume %x = 10:

Operation	**Operator**	**Example**	**Example's Final Value**
Addtion	+	10 + 10	20
Subtraction	-	20 - 10	10
Multiplication	*	10 * 10	100
Division	/	10 / 10	1
Modulous/Remainder	%	10 % 4	2
Additive Assignment	+=	%x += 10	20
Subtractive Assignment	-=	%x -= 10	0
Multiplicative Assignment	*=	%x *= 10	100
Divisive Assignment	/=	%x /= 10	1

These are the most basic of operations you can perform on a variable. This is basic arithmetic and you should know it already. A sample using these operators on variables would look like:

```
     %x = 10;
     %y = 10;
     Echo(%x + %y);
```

Notice that X equals 10 and Y equals 10, and you are echoing to the console 10 + 10, or 20. Easy, eh? Well, remember that these expressions aren't evaluated left-to-right as we read, there is a method by which mathematics are processes, call the 'Order of Operations' or in Computer Science, 'Operator Precedence'. Here is an example:

```
     %x = 10;
     %y = 10;
     %z = %x + %y * %y;
```

Z will equal 110, not 200. Operator Precedence can be modified with special operators called 'Precedence

Modifiers'. These are parentheses, '()'. An example would be:

```
     %x = 10;
     %y = 10;
     %z = (%x + %y) * %y;
```

Z will equal 200, not 110. These operators above are called 'Binary Operators', Binary means that it takes two 'things', or operands for it to work, whether they are constants or variables. There is another type of operators called 'Unary Operators'. These are Unary because they only take one operand to work. Here's a list of some basic ones, assume X is equal to 10:

Operation	**Operator**	**Example**	**%x's Final Value**
Incremental Operation	++	%x++	11
Decremental Operation	--	%x--	9

These unary operators can save you time in certain things, and will be used later on. There are also built-in functions that you can use. These are in the appendices.

3.3 Boolean Variables

George Boole invented a branch of mathematics call 'Boolean Algebra'. Unlike its number-based brethren, Boolean Algebra used 'true' or 'false'. I won't go into details, but boolean variables are perfect for computers because in a computer's CPU, there are tiny transistors that are either 'on' or 'off', or 'true' or 'false'. There are several operations that can be done with a boolean variable, here is a table, assume x is equal to true and y is equal to false:

Operation	**Operator**	**Example**	**%x's Final Value**
Not	!	!%x	false
And	&&	%x && %y	false
Or	||	%x || %y	true
Xor	None	%x XOR %y	true

There is no Logical Xor operator in TRIBES, but you'll be using XOR later on in the Bitwise chapter. Truth tables can be a big help, I've created some basic ones for you on the next page…

AND	**A**	**B**	**Result**
	TRUE	TRUE	TRUE
	TRUE	FALSE	FALSE
	FALSE	TRUE	FALSE
	FALSE	FALSE	FALSE

OR	**A**	**B**	**Result**
	TRUE	TRUE	TRUE
	TRUE	FALSE	TRUE
	FALSE	TRUE	TRUE
	FALSE	FALSE	FALSE

XOR	**A**	**B**	**Result**
	TRUE	TRUE	FALSE
	TRUE	FALSE	TRUE
	FALSE	TRUE	TRUE
	FALSE	FALSE	FALSE

NOT	**A**	**Result**
	TRUE	FALSE
	FALSE	TRUE

Booleans can help you more then you think, as you'll be using them extensively later on.

3.4 String Variables

You may be saying 'What the hell is a string?' Well, the easiest explanation would be that it's a bunch of characters (ex: 'A') 'strung' together. An example of a string would be "Hello" or 'I'm a TRIBES Coding Master!' You can create a string just like any other variable, except string constants must be surrounded in double-quotes. Example:

```
     %x = "Hello";
```

X will equal the string 'Hello'. Strings can also be combined, or concatenated. The concatenation operator is '@' or the less commonly used '$+', both do the same thing. Example:

```
     %x1 = "con";
     %x2 = "catenate";
     %x3 = %x1 @ %x2;
```

X3 will equal 'concatenate'. There are also special functions that you can use that will be explained later. String variables don't have a lot of built-in methods, however. There are several in the appendix that I've made myself, familiarize yourself with these as I'll be using them later on.

3.5 Object, Player, and Client Variables

Objects are things that are created inside the server. They could be a Repair Pack or a Particle Beam Weapon or a Player wearing a Heavy Armor. They all have an ID number to which TRIBES can locate them easily, this is what is held in a Object variable. They may seem just like Numerical Variables, but they can be used quite differently.

Player Variables are the same as object variables, except that they point to the Client's Player. This means that if I had a player variable of Client A, I'd have the Object ID of his current armor.

Client Variables are quite different. They point to a special number that points to a player's Client number. This client number is used by the engine for various things. If you're doing client-side scripting, the Client ID for the server is always 2048. The first client on a server is always 2049 and it goes up from there.

3.6 Property Variables

Objects, Clients, and Players all have something called 'Property Variables'. If you've ever used Visual Basic you know what I mean. A property variable is just like another variable, except it is 'paired' with another variable. These are odd, but can become quite useful. For example:

```
     %clientId.isAdmin = true;
```

These are frequently used by TRIBES Base to set a player to Admin. This variable 'isAdmin' is used in 'admin.cs' of the base mod quite a bit. You might say, 'Well, wouldn't a global variable be easier?' Well, it may be easier, but it wouldn't work as well. If you think that your code would look cleaner or more readable, use property variables instead of globals. Also, when you get into the chapter on datablocks, you’ll see that they also have property variables.

Chapter 4
Decision and Looping Statements

4.1 If-Else

Do you remember the boolean variable type earlier in the book? Well, you'll be using them all the time when you use statements that alter program flow. 'What's program flow?' You may be asking. Well, programs 'flow' or execute top-to-bottom, unless something alters its set course. The simplest of the statements to alter program flow is the If-Else statement. This is used like this:

```
     if(%bool)
     {
          //Do Some Stuff…
     }
     else
     {
          //Do Other Stuff…
     }
```

%bool is a boolean variable. The if() statement checks to see if the expression in it's parentheses evaluates to true, if it is, run the things inside it's brackets, {}. Otherwise just go on and skip everything inside the brackets. If the expression evaluates to false program execution hits the else statement and runs everything inside those brackets. You could also do something like this:

```
     if(%bool)
     {
          //Do Some Stuff…
     }
     else if(!%bool)
     {
          //Do Some Stuff…
     }
```

This is exactly the same thing as the first code block. You can even omit the else altogether. Example:

```
     if(%bool)
     {
          //Do Some Stuff…
     }
```

In the above code, the else statement is nothing. It would look like this:

```
	if(%bool)
	{
		//Do Some Stuff..
	} else {}
```

See? It should all be coming together now. Also, to shorten your work up a bit, if you only want to execute one statement if the expression is true, you can omit the brackets entirely like so:

```
	if(%bool)
		//Do One Thing…
```

Or…

```
	if(%bool)
		//Do One Thing…
	else
		//Do Another Thing…
```

Ok, now that you know the syntax, let's get into some more difficult things. So far, you've heard the term expression used quite a bit. The example was %bool, which is a really easy expression, it depends entirely upon the value of %bool. So, the expression evaluate to true if %bool is equal to true and vice-versa. It's not always that clear-cut. You can use several conditional operators to construct an expression. Here is a list, assume %x = 10 and %y = 20:

Operation	**Operator**	**Example**	**Evaluates to…**
Equal Conditional	==	%x == %y	false
Not Equal Conditional	!=	%x != %y	true
Less Than Conditional	<	%x < %y	true
Greater Than Conditional	>	%x > %y	false
Less Than Or Equal To Conditional	<=	%x <= %y	true
Greater Than Or Equal To Conditional	>=	%x >= %y	false

Also remember, that you can combine or negate expressions with the logical AND, NOT, and OR operators that were explained earlier. You can also use parentheses to modify the way the expression is interpreted. Here are some expression examples, assume %x = 10, %y = 20, and %z = 10:

```
	%x == %y		returns false
	%x == %z		returns true
	!(%x == %y)	returns true
	(%x != %y)	returns true
	(%x > %y)		returns false
```

Here are some more complicated ones:

```
     (%x != %y) && (%x == %z)           returns true
     (%x > %y) && (%x > (%z + 100))     returns false
```

Let’s examine this last expression ‘(%x > %y) && (%x > (%z + %y))’. We know that when you AND two expressions, they both have to be true for the result to be true. So, if the first expression is false, couldn’t we just stop there? We do, and so does the computer. This is called Short-Circuiting and makes it so if the first expression is false, then the second is never executed or evaluated. Well, now that we’ve got all that through, let’s go on.

4.2 While Loop

The while structure is an incredibly useful little thing. It allows you to repeat code any number of times, and it’s syntax is very similar to If-Else. Example:

```
     while(%bool)
     {
          //Do Something…
     }
```

You must remember though, that the expression used must be changed somehow in the loop, otherwise you’ll end up with something called an Infinite Loop. This occurs when the expression never evaluates to false so the loop can never stop. You might do something like this:

```
     %i = 1;
     while(%i < 5)
          %i++;
```

If you go though this, it will exit eventually because you are incrementing %i each time in the loop. Can you tell me how many times this loop will execute? If you said 4 you’re correct. If you said 5, that’s a common mistake so don’t fret. If you were to do the loop by hand you’d see that is goes from 1 to 4. When %i is equal to 5, the loop exits. You can use any expression or boolean variable, just make sure they it exits eventually.

4.3 For Loop

The For Loop structure is my favorite loop, as I think it’s the most useful. The for loop consists of 3 parts, besides the body. These are the Initialization, Evaluation, and Incremental parts. Any of these can be omitted, but if you use none of them, you end up with an infinite loop and you need to use the ‘break’ statement inside the loop. Here are two examples:

```
     for(%i = 1; %i < 5; %i++)
     {
          //Do Some Stuff
     }

     %i = 1;
     for(;;)
     {
          if(%i < 5)
               break;

          //Do Some Stuff

          %i++;
     }
```

They both do the exact same thing. There is also another special statement you can use called ‘continue’. This ends the current execution of the loop and goes on to the beginning again. Here is an example:

```
     for(%i = 1; %i < 5; %i++)
     {
          if(%i == 1)
               continue;
          //Do Some Stuff
     }
```

The first run of the loop, %i will equal to 1 and execution will stop there, and go back to the beginning, and %i well then equal to 2 and everything will go on as normal. These special keywords can be used in any loop, whether While or For.

Chapter 5
Functions and Datablocks

5.1 Functions

Functions are the 'bread-and-butter' of TRIBES Script. Functions can call themselves, other functions, or built-in functions. A CS file itself can hold program statements, or call a function, but typically you only run a CS file once, and that's it. A function is denoted by the 'function' keyword, an identifier, an argument list, and a body. Here is an example function:

```
function test(%var)
{
     %var++;
     echo(%var);
}
```

As you can see, this code block has the function keyword, an identifier, an argument list, and a body that is surrounded with brackets. Functions can return a value with the 'return' keyword. An example a return function would be:

```
function addOne(%var)
{
     return %var + 1;
}
```

When a function calls itself, it's called a recursive function. Here is an example of a recursive function:

```
function recursive(%var)
{
     if(%var <= 0)
          return;
     return %var + recursive(%var – 1);
}
```

You may be saying 'What the hell is all that?' Recursive functions are useful, but can be complicated. If you want a good look at recursive loops go take a Computer Science course. This recursive loop here will add all integers starting with the value passed into it to zero. An iterative loop of the same might look like:

```
function addAllFrom(%var)
{
     for(%i = %var; %i <= 0; %i--)
     {
          %tmp += %i;
     }
     return %tmp;
}
```

You might be tempted to do something like this:

```
function addOne(%var)
{
     %var++;
}
```

This will not do as expected, because variables are 'copied' from function to function. So, do NOT do this.

There are several built-in functions, you already know one 'echo()'. Originally, 'www.tribesplayers.com' had a list of most of these, but this website was shutdown quite a while ago. Now, you can find these in various places. I've included an index of all the functions I know about in the appendix that are not in the official documents. You can get a copy of the official documents from 'http://www.annihilation.info/'. There is also a special type of function called a remote function, this function can be called from the client and executed on the server, or vice-versa. Here is an example:

```
//On Server…
function remoteFunction(%clientId, %msg)
{
     echo(%msg);
}

//Call from Client…
remoteEval(2048, Function, "Hello World!");
```

If the client called that and the remoteFunction was on the server then "Hello World!" would be echoed on the server. Also, there are special functions that allow you to run a function from a string, or run a function in the near future. Here are some examples:

```
     eval(“function();”);
     schedule(“function();”, 1);
```

The first statement will run function() just as if you were directly calling it. The second statement will call function() 1 second from now.

5.2 Datablocks

Datablocks are basically Object Descriptions, or Items, Weapons, Armors, etc. You can find various examples of datablocks inside the base mod scripts. A sample datablock for a plant object might look like this:

```
     StaticShapeData Plant
     {
          shapeFile = "plant1";
          debrisId = defaultDebrisSmall;
          maxDamage = 0.4;
          description = "Plant";
     };
```

There are many properties and such that I can’t list them all here, but if you look at existing objects, items, and such you should get a pretty good idea on how to construct your own.

Chapter 6
Arrays

6.1 Arrays / Subscripted Variables

Arrays and Subscripted Variables are essentially the same. However, one is shorter than the other so I'll be using 'Arrays'. An Array in TRIBES is a bit different then a conventional programming language's idea of an array. In TRIBES, an array is basically just an add-on to the variable's identifier. Here is an example:

```
     $Array[0] = "Hello";
     $Array0 = "World";
```

You may say, 'The first variable is "Hello" and the second is "World".' In a different language this would be true, but in TRIBES, this isn't the case; they both equal "World". You aren't restricted to only using numerical values, you can also use Booleans, Strings, or Datablock names. Here is another few examples:

```
     $Array[true] = 0;
     $Arraytrue = 1;
     //Both equal to 1

     $Array["Hey"] = 0;
     $ArrayHey = 1;
     //Both equal to 1

     $Array[Blaster] = 0;
     $ArrayBlaster = 1;
     //Both equal to 1
```

Awesome stuff! Also, you can use variables as the subscript. Example:

```
     %var = 0;
     $Array[%var] = 1;
     $Array0 = 0;
     //Both equal to 0
```

Now, I should also mention that these 'arrays' are *not* true arrays. The obvious difference is that you can use subscripts other then integers and they aren't sequential. The less-than obvious reason is that they don't share sequential memory addresses.

6.2 Array Techniques

Arrays are very powerful and can be used for a database, or just for storing variables in a neat fashion.

To loop though an array correctly, you need to use numbers, as these are the easiest to work with. Here is an example:

```
     for(%i = 0; $Array[%i] != “”; %i++)
     {
          echo($Array[%i]);
     }
```

This will loop though the entire Array variable and echo every element to the server. As a little tip, when ever a variable is not initialized, but used in an expression, it is either 0, “”, or false, depending on the expression. As an improvement to the above code, you could do:

```
     for(%i = 0; (%current = $Array[%i]) != “”; %i++)
     {
          echo(%current);
     }
```

This adds a little assignment operation into the loop, so %current always contains the value of the current array element. You can also loop backwards, only you need to know where to start. Let’s assume that $Array has 30 elements.

```
     for(%i = 30; %i >= 0; %i++)
     {
          echo(%current);
     }
```

Instead of looping forward like the first examples, if you know the length, you can also do something like this:

```
     for(%i = 0; %i <= 30; %i++)
     {
          echo(%current);
     }
```

There are also some sorting and searching techniques…

Chapter 7
Sorting and Searching

7.1 Linear Search

The linear search is probably the easiest of searches to perform. Here is the code for a linear search:

```
function linearSearch(%search)
{
     %found = -1;
     for(%i = 0; (%current = $Array[%i]) != ""; %i++)
     {
          if(%current == %search)
               %found = %i;
     }
     return %found;
}
```

This will work well enough, but there is a problem: the function will loop though *every* element, even if it's already found what it's looking for. That's like saying 'Well, I found my car keys under the sofa, but I better look in the bathroom incase they are there.'! Doesn't make sense does it? Well let's fix it:

```
function linearSearch(%search)
{
     for(%i = 0; (%current = $Array[%i]) != ""; %i++)
     {
          if(%current == %search)
               return %i;
     }
     return -1;
}
```

As you can see, this returns the value as soon as it's found, giving a good-sized efficiency bonus.

7.2 Numeric Sorting

To sort an array, you first must be able to swap an array's variables. Here is a sample function that I'll be using:

```
function swap(%x, %y)
{
     %tmp = $Array[%x];
     $Array[%x] = $Array[%y];
     $Array[%y] = %tmp;
}
```

Simple, right? Ok, now that we have our swap function, we need to make the sort function. Here is an example:

```
function numberSort()
{
     for(%i = 0; $Array[%i] != “”; %i++)
     {
          %swap = %i;
          for(%j = 0; (%compare = $Array[%j]) != “”; %j++)
          {
               if ($Array[%swap] > %compare)
               {
                    %swap = %j;
               }
          }
          swap(%i, %swap);
     }
}
```

Now, this may look unwieldy, but its actually quite simple! This type of sort is called a ‘Bubble Sort’. What it does is run though the entire array once, and then each time checking if the current value of the array element ‘%swap’ is greater than the ‘%compare’ element. If it is, set it to be swapped, otherwise continue on. When the second for loop has finished, it swaps the values and goes on. This can become very CPU intensive as you might have guessed, so there are better ways to sort. But, I’m not going to go into them, if you want some further study, you can look up some Java tutorials or join a class.

7.3 Alphabetic/Lexicographic Sorting

Well, we can sort numbers, great! However, what if we want to sort a list of player names? Well, we would use the same algorithm, but instead of checking if the two variables are greater than each other, we are going to check to see if the string itself is ‘above’ or ‘below’ the other string. I’ve included my own method for this in the appendix. Here is some sample code of the sort:

```
function alphaSort()
{
     for(%i = 0; $Array[%i] != “”; %i++)
     {
          %swap = %i;
          for(%j = 0; (%compare = $Array[%j]) != “”; %j++)
          {
```

```
     if(String::IsGreaterThan($Array[%swap], %compare))
               {
                    %swap = %j;
               }
          }
          swap(%i, %swap);
     }
}
```

Chapter 8
Bitwise

8.1 Bitwise Theory

I've gained a lot of this information from experience and some very good tutorials over at www.gamedev.net. If there are similarities it is because I used these to help me write this section, as I'm not a teacher. A number is a very abstract thing. Unlike physical objects, which are easily recognizable, a number can be represented in any number of ways. The representation we are use is called decimal, or base 10. The first term is pretty familiar, but if you're reading this theory, the second term may be new to you. To see why we use the term "base 10", let's take a look at a number, say 4232. Read aloud, this is 'four thousand, two hundred, thirty-two'. We hear numbers like that so often that it's not immediately obvious, but this sounds a lot like a formula:

4232 = (4 * 1000) + (2 * 100) + (3 * 10) + (2 * 1)

Or, if we write it another way, we see that a decimal number is actually the sum of its digits multiplied by successive powers of 10:

$4232 = (4 * 10^3) + (2 * 10^2) + (3 * 10^1) + (2 * 10^0)$

Do you now see why the term "base 10" is used to describe the way we usually write numbers? The obvious question to ask now is, "Why do we have to use 10 as the base?" We don't! Any positive integer greater than two can be used as the base. Of course, if we tried to use a "base 1" number system, the only thing we could write would be strings of zeroes. So to analyze further, if we take our last formula, and replace the 10 with a generic base B, then we have the representation for a number in any base. Now, computers deal with binary, or base 2. This is because of the nature of CPUs, the transistors inside them are either 'On' or 'Off', '0' or '1'. For example, consider the binary number 100101. The value of this number is:

$100101 = (1 * 2^5) + (0 * 2^4) + (0 * 2^3) + (1 * 2^2) + (0 * 2^1) + (1 * 2^0) = 32 + 4 + 1 = 37$

The hexadecimal number system uses base 16, which means that there are sixteen digits that can be used. Of course, we are accustomed to having ten, 0 through 9. In hexadecimal, however, the character A has a value of 10, B has a value of 11, and so on to F, which has a value of 15. Let's see an example of this:

$3FC = (3 * 16^2) + (F * 16^1) + (C * 16^0) = (3 * 16^2) + (15 * 16^1) + (12 * 16^0) = 768 + 240 + 12 = 1020$

The reason hexadecimal is so frequently used in programming is that it's very easy to translate between hexadecimal and binary. Not so when converting from decimal to binary. The reason it's so easy to convert between base 16 and base 2 is because 16 is a power of two. $16 = 2^4$. Why is this significant? Well, a group of four binary digits can take exactly 2^4 values, which means that each hexadecimal digit corresponds to exactly four binary digits. Since ten is not a power of two, the conversion is harder in decimal. So when programmers want to use a specific binary number, they write it in hexadecimal. In TRIBES Script, you can recognize a hexadecimal number because it is always prefixed by "0x". For example, this statement assigns the value 1020 to a variable, by using its hexadecimal equivalent:

```
     %hexValue = 0x3FC;
```

There is no such prefix that will allow you to write a binary number directly, which is why hexadecimal is used. The following table shows the binary equivalents for each of the sixteen hex digits.

Decimal	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Hexadecimal	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
Binary	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111

We can use this table to show a quick example, finding the binary equivalent of the hex number 3FC. Simply convert each hex digit to its binary equivalent, and you're all done:

3FC = 0011 1111 1100 = 1111111100

In binary, we can drop leading zeros just like in base 10. Onto the operators!

8.2 Bitwise Operations

Bitwise has almost the same operations as boolean variables, except they are slightly different. Instead of two expressions, it’s two bits. Here are the truth tables:

AND	**A**	**B**	**Result**
	0	0	0
	0	1	1
	1	0	1
	1	1	1

OR	**A**	**B**	**Result**
	0	0	0
	0	1	0
	1	0	0
	1	1	1

XOR	**A**	**B**	**Result**
	0	0	1
	0	1	0
	1	0	0
	1	1	1

These all have a operator for their respective operation, here is a list:

Operation	**Operator**
Bitwise AND	&
Bitwise OR	|
Bitwise XOR	^
Bitwise Complement	~
Bitwise AND Assignment	&=
Bitwise OR Assignment	|=

There are also something called bitwise shifts, these are useful because you can 'shift' all the bits in a number left or right a certain number of times. Shifting left N times is the same as multiplication by 2 to the power of N. The opposite goes for shifting right, where shifting right N times is the same as division by 2 to the power of N. This is illustrated below:

```
%x = %y * 8;
%x = %y << 3;

%x = %y * 64;
%x = %y << 6;

%x = %y * 32768;
%x = %y << 15;
```

Pretty cool, isn't it? This is a fast way to accomplish multiplication by powers of two, using only a bitwise shift. For reference, here is a list of all shift operators:

Operation	**Operator**
Shift Left	<<
Shift Right	>>
Shift Left Assignment	<<=
Shift Right Assignment	>>=

Well, now that all that is out of the way, let's look at what we can use this for!

8.3 Uses for Bitwise

The most common use for bitwise is to only use one variable for an almost infinite amount of 'attributes'. Here is an example:

```
$Expert = 1 << 1; //2
$Admin  = 1 << 2; //4
$Banned = 1 << 3; //8
$Coder  = 1 << 4; //16

function isPlayer(%property)
{
     return (($PlayerProp & %property) > 0);
}

function addProperty(%property)
{
     $PlayerProp |= %property;
}

function removeProperty(%property)
{
     $PlayerProp ^= %property;
}

addProperty($Expert | $Coder);
addProperty($Admin);
if(isPlayer($Coder))
     echo("Player is a coder.");
else
     echo("Player is not a coder.");
```

If you were to run this code snippet, you'd find that 'Player is a coder.' would be displayed on the console.

Really cool, eh? ModX uses this system to store info about it’s player’s profiles, and I’m sure you can think of other uses for this.

Chapter 9
Closing Words

9.1 Last Words

Well, I hope I have enlightened you to the ways of the TRIBES Script Coder. However, don't consider this the only thing you should ever read. Computer Science books and tutorials are great. Should you need more help, you can always go to great sites like http://www.annihilation.info/, http://modx.ath.cx:1337/, or even http://www.gamedev.net/. Remember, if you ask questions on a forum, make the post intelligible and list what you've done, and what you need.

Appendices and References

Appendix

The following pages contain various functions that I encourage you to use in your own modifications. Of course, remember to put a side-note mentioning who you learned to code from ;)

```
function String::IsWhiteSpace(%string)
{
    %slen = String::Len(%string);
    %IsWhiteSpace = true;

      for(%i = 0; %i < %slen; %i++)
      {
        //
        if(String::getSubStr(%string, %i, 1) == " ")
            %IsWhiteSpace = true;
        else
            %IsWhiteSpace = false;
    }

    return %IsWhiteSpace;
}

function String::IsGreaterThan(%string, %compare)
{
    %slen = String::Len(%string);
    %clen = String::Len(%compare);

    %string = String::MakeCaps(%string);
    %compare = String::MakeCaps(%compare);

      for(%i = 0; %i < %slen; %i++)
      {
            if(%clen <= %i)
                  return false;
            if(String::CharacterValue(String::charAt(%string, %i)) <
String::CharacterValue(String::charAt(%compare, %i)))
                  return true;
            if(String::CharacterValue(String::charAt(%string, %i)) >
String::CharacterValue(String::charAt(%compare, %i)))
                  continue;
            if(String::CharacterValue(String::charAt(%string, %i)) ==
String::CharacterValue(String::charAt(%compare, %i)))
                  continue;
      }
      return false;
}

function String::IsLessThan(%string, %compare)
{
    return !String::IsGreaterThan(%string, %compare);
}

function String::Reverse(%string)
{
    %slen = String::Len(%string);
    %outString = "";
    for(%i = %slen + 1; %i >= 0; %i--)
    {
        %outString = %outString @ String::CharAt(%string, %i);
    }
    return %outString;
}

function String::MakeCaps(%string)
{
   %norm = "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ";
   %capp = "ABCDEFGHIJKLMNOPQRSTUVWXYZABCDEFGHIJKLMNOPQRSTUVWXYZ";
   for(%i = 0; %i < 52; %i++)
   {
      %rem = String::getSubStr(%norm, %i, 1);
      %new = String::getSubStr(%capp, %i, 1);
      %string = String::Replace(%string, %rem, %new);
   }
   return %string;
}

function String::MakeLower(%string)
{
   %norm = "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ";
   %low  = "abcdefghijklmnopqrstuvwxyzabcdefghijklmnopqrstuvwxyz";
   for(%i = 0; %i < 52; %i++)
   {
      %rem = String::getSubStr(%norm, %i, 1);
      %new = String::getSubStr(%low, %i, 1);
      %string = String::Replace(%string, %rem, %new);
   }
   return %string;
}

function String::IsAlpha(%string)
{
   %norm = "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ";
   %slen = String::Len(%string);
   for(%i = 0; %i < %slen; %i++)
   {
      %ch = String::CharAt(%i);
      for(%f = 0; %f < 52; %f++)
      {
          if(%ch == String::CharAt(%f))
          {
              %out = %out @ %ch;
          }
      }
   }
   return (%string == %out);
}

function String::Len(%string)
{
      while(String::getSubStr(%string, %len, 1) != "")
            %len++;
      return %len;
}

function String::Replace(%string, %search, %replace)
{
      %len = String::Len(%search);
      for (%i = 0; (%char = String::getSubStr(%string, %i, %len)) != "";
%i++)
      {
            if (%char @ "s" == %search @ "s")
                  %string = String::getSubStr(%string, 0, %i) @ %replace
@ String::getSubStr(%string, %i + %len, 255);
      }
      return %string;
}
```

```
function String::Contains(%string, %search)
{
      %len = String::Len(%search);
      for (%i = 0; (%char = String::getSubStr(%string, %i, %len)) != "";
%i++)
      {
            if (%char @ "s" == %search @ "s")
                  return true;
      }
      return false;
}

function String::indexOf(%string, %search)
{
      %len = String::Len(%search);
      for (%i = 0; (%char = String::getSubStr(%string, %i, %len)) != "";
%i++)
      {
            if (%char @ "s" == %search @ "s")
                  return %i;
      }
      return -1;
}

function String::Left(%string, %start)
{
      %len = 1;
    %stringFinal = "";
      for (%i = 0 + %start; (%char = String::getSubStr(%string, %i, %
len)) != ""; %i++)
      {
        %stringTemp = %stringFinal;
        %stringFinal = %stringTemp @ %char;
    }

    if(%stringFinal != "")
          return %stringFinal;
    return -1;
}

function String::Right(%string, %start)
{
      %len = 1;
    %stringFinal = "";
      for (%i = 0 + %start; (%char = String::getSubStr(%string, %i, %
len)) != ""; %i--)
      {
        %stringTemp = %stringFinal;
        %stringFinal = %char @ %stringTemp;
    }

    if(%stringFinal != "")
          return %stringFinal;
    return -1;
}

function String::Mid(%string, %start, %stop)
{
    return String::getSubStr(%string, %start, %stop);
}
```

```
function String::ParseIP(%address)
{
	if(String::getSubStr(%address,0,8) == "LOOPBACK")
		return "LOOPBACK";

	%ipCut = String::getSubStr(%address,3,20);
	while(String::getSubStr(%ipCut,%len,1) != ":" && %len < 20)
		%len++;
	%sub = String::getSubStr(%ipCut,0,%len);
	return %sub;
}

$OmitList[0] = "a";
$OmitList[1] = "b";
$OmitList[2] = "c";
$OmitList[3] = "d";
$OmitList[4] = "e";
$OmitList[5] = "f";
$OmitList[6] = "g";
$OmitList[7] = "h";
$OmitList[8] = "i";
$OmitList[9] = "j";
$OmitList[10] = "k";
$OmitList[11] = "l";
$OmitList[12] = "m";
$OmitList[13] = "n";
$OmitList[14] = "o";
$OmitList[15] = "p";
$OmitList[16] = "q";
$OmitList[17] = "r";
$OmitList[18] = "s";
$OmitList[19] = "t";
$OmitList[20] = "u";
$OmitList[21] = "v";
$OmitList[22] = "w";
$OmitList[23] = "x";
$OmitList[24] = "y";
$OmitList[25] = "z";

$OmitList[26] = "A";
$OmitList[27] = "B";
$OmitList[28] = "C";
$OmitList[29] = "D";
$OmitList[30] = "E";
$OmitList[31] = "F";
$OmitList[32] = "G";
$OmitList[33] = "H";
$OmitList[34] = "I";
$OmitList[35] = "J";
$OmitList[36] = "K";
$OmitList[37] = "L";
$OmitList[38] = "M";
$OmitList[39] = "N";
$OmitList[40] = "O";
$OmitList[41] = "P";
$OmitList[42] = "Q";
$OmitList[43] = "R";
$OmitList[44] = "S";
$OmitList[45] = "T";
$OmitList[46] = "U";
$OmitList[47] = "V";
```

```
$OmitList[48] = "W";
$OmitList[49] = "X";
$OmitList[50] = "Y";
$OmitList[51] = "Z";

$OmitList[52] = "~";
$OmitList[53] = "`";
$OmitList[54] = "!";
$OmitList[55] = "@";
$OmitList[56] = "#";
$OmitList[57] = "$";
$OmitList[58] = "%";
$OmitList[59] = "^";
$OmitList[60] = "&";
$OmitList[61] = "*";
$OmitList[62] = "(";
$OmitList[63] = ")";
$OmitList[64] = " "; // -
$OmitList[65] = "_";
$OmitList[66] = "=";
$OmitList[67] = "+";
$OmitList[68] = "[";
$OmitList[69] = "{";
$OmitList[70] = "]";
$OmitList[71] = "}";
$OmitList[72] = "\\";
$OmitList[73] = "|";
$OmitList[74] = "/";
$OmitList[75] = "?";
$OmitList[76] = " "; // .
$OmitList[77] = ">";
$OmitList[78] = ",";
$OmitList[79] = "<";
$OmitList[80] = " ";

function String::NumericParse(%string)
{
    %replace = "";
    %len = 1;
    for (%j = 0; (%search = $OmitList[%j]) != "" && %j < 100; %j++)
      {
      for (%i = 0; (%char = String::getSubStr(%string, %i, %len)) != ""
&& %i < 300; %i++)
      {
            if (%search @ "s" == %char @ "s")
                  %string = String::getSubStr(%string, 0, %i) @ %replace
@ String::getSubStr(%string, %i + %len, 255);
        }
    }
    return %string;
}

function String::Occurances(%string, %search)
{
      %len = String::Len(%search);
    %o = 0;
    for (%i = 0; (%char = String::getSubStr(%string, %i, %len)) != ""; %
i++)
      {
            if (%char @ "s" == %search @ "s")
                  %o++;
      }
      return %o;
}

function String::CheckHexCrash(%string) //TRUE = Something is wrong!
{
    %size = 0;

    %slen = String::Len(%string);
    %xoc = String::Occurances(escapeString(%string), "\\x");
    %toc = String::Occurances(%string, "\n");
    %noc = String::Occurances(%string, "\t");

    %size += (%slen - (%xoc + %toc + %noc));
    %size += (%xoc * 4);
    %size += (%toc * 4);
    %size += (%noc * 4);
    if(%size > 510)
    {
        return true;
    }
    return false;
}

$CharacterList[0] = "\x00";
$CharacterList[1] = "\x01";
$CharacterList[2] = "\x02";
$CharacterList[3] = "\x03";
$CharacterList[4] = "\x04";
$CharacterList[5] = "\x05";
$CharacterList[6] = "\x06";
$CharacterList[7] = "\x07";
$CharacterList[8] = "\x08";
$CharacterList[9] = "\x09";
$CharacterList[10] = "\x0A";
$CharacterList[11] = "\x0B";
$CharacterList[12] = "\x0C";
$CharacterList[13] = "\x0D";
$CharacterList[14] = "\x0E";
$CharacterList[15] = "\x0F";
$CharacterList[16] = "\x10";
$CharacterList[17] = "\x11";
$CharacterList[18] = "\x12";
$CharacterList[19] = "\x13";
$CharacterList[20] = "\x14";
$CharacterList[21] = "\x15";
$CharacterList[22] = "\x16";
$CharacterList[23] = "\x17";
$CharacterList[24] = "\x18";
$CharacterList[25] = "\x19";
$CharacterList[26] = "\x1A";
$CharacterList[27] = "\x1B";
$CharacterList[28] = "\x1C";
$CharacterList[29] = "\x1D";
$CharacterList[30] = "\x1E";
$CharacterList[31] = "\x1F";
$CharacterList[32] = "\x20";
$CharacterList[33] = "\x21";
$CharacterList[34] = "\x22";
$CharacterList[35] = "\x23";
$CharacterList[36] = "\x24";
```

```
$CharacterList[37] = "\x25";
$CharacterList[38] = "\x26";
$CharacterList[39] = "\x27";
$CharacterList[40] = "\x28";
$CharacterList[41] = "\x29";
$CharacterList[42] = "\x2A";
$CharacterList[43] = "\x2B";
$CharacterList[44] = "\x2C";
$CharacterList[45] = "\x2D";
$CharacterList[46] = "\x2E";
$CharacterList[47] = "\x2F";
$CharacterList[48] = "\x30";
$CharacterList[49] = "\x31";
$CharacterList[50] = "\x32";
$CharacterList[51] = "\x33";
$CharacterList[52] = "\x34";
$CharacterList[53] = "\x35";
$CharacterList[54] = "\x36";
$CharacterList[55] = "\x37";
$CharacterList[56] = "\x38";
$CharacterList[57] = "\x39";
$CharacterList[58] = "\x3A";
$CharacterList[59] = "\x3B";
$CharacterList[60] = "\x3C";
$CharacterList[61] = "\x3D";
$CharacterList[62] = "\x3E";
$CharacterList[63] = "\x3F";
$CharacterList[64] = "\x40";
$CharacterList[65] = "\x41";
$CharacterList[66] = "\x42";
$CharacterList[67] = "\x43";
$CharacterList[68] = "\x44";
$CharacterList[69] = "\x45";
$CharacterList[70] = "\x46";
$CharacterList[71] = "\x47";
$CharacterList[72] = "\x48";
$CharacterList[73] = "\x49";
$CharacterList[74] = "\x4A";
$CharacterList[75] = "\x4B";
$CharacterList[76] = "\x4C";
$CharacterList[77] = "\x4D";
$CharacterList[78] = "\x4E";
$CharacterList[79] = "\x4F";
$CharacterList[80] = "\x50";
$CharacterList[81] = "\x51";
$CharacterList[82] = "\x52";
$CharacterList[83] = "\x53";
$CharacterList[84] = "\x54";
$CharacterList[85] = "\x55";
$CharacterList[86] = "\x56";
$CharacterList[87] = "\x57";
$CharacterList[88] = "\x58";
$CharacterList[89] = "\x59";
$CharacterList[90] = "\x5A";
$CharacterList[91] = "\x5B";
$CharacterList[92] = "\x5C";
$CharacterList[93] = "\x5D";
$CharacterList[94] = "\x5E";
$CharacterList[95] = "\x5F";
$CharacterList[96] = "\x60";
$CharacterList[97] = "\x61";
```

```
$CharacterList[98] = "\x62";
$CharacterList[99] = "\x63";
$CharacterList[100] = "\x64";
$CharacterList[101] = "\x65";
$CharacterList[102] = "\x66";
$CharacterList[103] = "\x67";
$CharacterList[104] = "\x68";
$CharacterList[105] = "\x69";
$CharacterList[106] = "\x6A";
$CharacterList[107] = "\x6B";
$CharacterList[108] = "\x6C";
$CharacterList[109] = "\x6D";
$CharacterList[110] = "\x6E";
$CharacterList[111] = "\x6F";
$CharacterList[112] = "\x70";
$CharacterList[113] = "\x71";
$CharacterList[114] = "\x72";
$CharacterList[115] = "\x73";
$CharacterList[116] = "\x74";
$CharacterList[117] = "\x75";
$CharacterList[118] = "\x76";
$CharacterList[119] = "\x77";
$CharacterList[120] = "\x78";
$CharacterList[121] = "\x79";
$CharacterList[122] = "\x7A";
$CharacterList[123] = "\x7B";
$CharacterList[124] = "\x7C";
$CharacterList[125] = "\x7D";
$CharacterList[126] = "\x7E";
$CharacterList[127] = "\x7F";
$CharacterList[128] = "\x80";
$CharacterList[129] = "\x81";
$CharacterList[130] = "\x82";
$CharacterList[131] = "\x83";
$CharacterList[132] = "\x84";
$CharacterList[133] = "\x85";
$CharacterList[134] = "\x86";
$CharacterList[135] = "\x87";
$CharacterList[136] = "\x88";
$CharacterList[137] = "\x89";
$CharacterList[138] = "\x8A";
$CharacterList[139] = "\x8B";
$CharacterList[140] = "\x8C";
$CharacterList[141] = "\x8D";
$CharacterList[142] = "\x8E";
$CharacterList[143] = "\x8F";
$CharacterList[144] = "\x90";
$CharacterList[145] = "\x91";
$CharacterList[146] = "\x92";
$CharacterList[147] = "\x93";
$CharacterList[148] = "\x94";
$CharacterList[149] = "\x95";
$CharacterList[150] = "\x96";
$CharacterList[151] = "\x97";
$CharacterList[152] = "\x98";
$CharacterList[153] = "\x99";
$CharacterList[154] = "\x9A";
$CharacterList[155] = "\x9B";
$CharacterList[156] = "\x9C";
$CharacterList[157] = "\x9D";
$CharacterList[158] = "\x9E";
```

```
$CharacterList[159] = "\x9F";
$CharacterList[160] = "\xA0";
$CharacterList[161] = "\xA1";
$CharacterList[162] = "\xA2";
$CharacterList[163] = "\xA3";
$CharacterList[164] = "\xA4";
$CharacterList[165] = "\xA5";
$CharacterList[166] = "\xA6";
$CharacterList[167] = "\xA7";
$CharacterList[168] = "\xA8";
$CharacterList[169] = "\xA9";
$CharacterList[170] = "\xAA";
$CharacterList[171] = "\xAB";
$CharacterList[172] = "\xAC";
$CharacterList[173] = "\xAD";
$CharacterList[174] = "\xAE";
$CharacterList[175] = "\xAF";
$CharacterList[176] = "\xB0";
$CharacterList[177] = "\xB1";
$CharacterList[178] = "\xB2";
$CharacterList[179] = "\xB3";
$CharacterList[180] = "\xB4";
$CharacterList[181] = "\xB5";
$CharacterList[182] = "\xB6";
$CharacterList[183] = "\xB7";
$CharacterList[184] = "\xB8";
$CharacterList[185] = "\xB9";
$CharacterList[186] = "\xBA";
$CharacterList[187] = "\xBB";
$CharacterList[188] = "\xBC";
$CharacterList[189] = "\xBD";
$CharacterList[190] = "\xBE";
$CharacterList[191] = "\xBF";
$CharacterList[192] = "\xC0";
$CharacterList[193] = "\xC1";
$CharacterList[194] = "\xC2";
$CharacterList[195] = "\xC3";
$CharacterList[196] = "\xC4";
$CharacterList[197] = "\xC5";
$CharacterList[198] = "\xC6";
$CharacterList[199] = "\xC7";
$CharacterList[200] = "\xC8";
$CharacterList[201] = "\xC9";
$CharacterList[202] = "\xCA";
$CharacterList[203] = "\xCB";
$CharacterList[204] = "\xCC";
$CharacterList[205] = "\xCD";
$CharacterList[206] = "\xCE";
$CharacterList[207] = "\xCF";
$CharacterList[208] = "\xD0";
$CharacterList[209] = "\xD1";
$CharacterList[210] = "\xD2";
$CharacterList[211] = "\xD3";
$CharacterList[212] = "\xD4";
$CharacterList[213] = "\xD5";
$CharacterList[214] = "\xD6";
$CharacterList[215] = "\xD7";
$CharacterList[216] = "\xD8";
$CharacterList[217] = "\xD9";
$CharacterList[218] = "\xDA";
$CharacterList[219] = "\xDB";
```

```
$CharacterList[220] = "\xDC";
$CharacterList[221] = "\xDD";
$CharacterList[222] = "\xDE";
$CharacterList[223] = "\xDF";
$CharacterList[224] = "\xE0";
$CharacterList[225] = "\xE1";
$CharacterList[226] = "\xE2";
$CharacterList[227] = "\xE3";
$CharacterList[228] = "\xE4";
$CharacterList[229] = "\xE5";
$CharacterList[230] = "\xE6";
$CharacterList[231] = "\xE7";
$CharacterList[232] = "\xE8";
$CharacterList[233] = "\xE9";
$CharacterList[234] = "\xEA";
$CharacterList[235] = "\xEB";
$CharacterList[236] = "\xEC";
$CharacterList[237] = "\xED";
$CharacterList[238] = "\xEE";
$CharacterList[239] = "\xEF";
$CharacterList[240] = "\xF0";
$CharacterList[241] = "\xF1";
$CharacterList[242] = "\xF2";
$CharacterList[243] = "\xF3";
$CharacterList[244] = "\xF4";
$CharacterList[245] = "\xF5";
$CharacterList[246] = "\xF6";
$CharacterList[247] = "\xF7";
$CharacterList[248] = "\xF8";
$CharacterList[249] = "\xF9";
$CharacterList[250] = "\xFA";
$CharacterList[251] = "\xFB";
$CharacterList[252] = "\xFC";
$CharacterList[253] = "\xFD";
$CharacterList[254] = "\xFE";
$CharacterList[255] = "\xFF";
$CharacterList[256] = "\t";
$CharacterList[257] = "\n";
$CharacterList[258] = "END";

function String::CharacterValue(%char)
{
    %len = String::Len(%char);
    for (%j = 0; (%search = $CharacterList[%j]) != "END"; %j++)
      {
        if (%search @ "s" == String::charAt(%char, 0) @ "s")
        {
            return %j;
        }
    }
    return -1;
}

function String::fromCharCode(%code)
{
    for (%j = 0; (%search = $CharacterList[%j]) != "END"; %j++)
      {
        if (%j == %code)
            return %search;
    }
    return "";
}

function String::charAt(%string, %index)
{
    return String::getSubStr(%string, %index, 1);
}

function String::charValueAt(%string, %index)
{
    return String::CharacterValue(String::charAt(%string, %index));
}

function String::trim(%string)
{
    if(String::Replace(%string, " ", "") == "")
        return "";
    %slen = String::Len(%string);
    for(%i = 0; %i < %slen; %i++)
    {
        if(String::charAt(%string, %i) == " ")
            %ftrim++;
        else
            break;
    }
    for(%i = %slen - 1; %i >= 0; %i--)
    {
        if(String::charAt(%string, %i) == " ")
            %btrim++;
        else
            break;
    }
    return String::Mid(%string, %ftrim, %slen - %btrim);
}
```

```
$Math::PI = "3.14159265358979323";

function Math::rad2deg(%radians)
{
    return %radians * (180 / $Math::PI);
}

function Math::deg2rad(%degrees)
{
    return %degrees * ($Math::PI / 180);
}

function Math::roundDown(%delta)
{
    return floor(%delta - 0.01);
}

function Math::roundUp(%delta)
{
    return ceil(%delta + 0.01);
}

function Math::randomInt(%max)
{
    return floor(getRandom() * (%max - 0.01));
}

function Math::sin(%theta)
{
    if(%theta == "NaN")
        return;
      return (%theta - (pow(%theta,3)/6) + (pow(%theta,5)/120) - (pow(%
theta,7)/5040) + (pow(%theta,9)/362880) - (pow(%theta,11)/39916800));
}

function Math::cos(%theta)
{
    if(%theta == "NaN")
        return;
      return (1 - (pow(%theta,2)/2) + (pow(%theta,4)/24) - (pow(%
theta,6)/720) + (pow(%theta,8)/40320) - (pow(%theta,10)/3628800));
}

function Math::tan(%theta)
{
    if(%theta == "NaN")
        return;
    return Sin(%theta) / Cos(%theta);
}

function Math::absolute(%delta)
{
    return sqrt(pow(%delta,2));
}

function Math::isNaN(%number)
{
    if(%number == "NaN" || String::NumericParse(%number) != %number)
        return true;
    return false;
}
```

```
function Vector::rotate(%vec,%rot)
{
      %pi = $Math::PI;
      %rot3= getWord(%rot,2);
      for(%i = 0; %rot3 >= %pi*2; %i++) %rot3 = %rot3 - %pi*2;
      if (%rot3 > %pi) %rot3 = %rot3 - %pi*2;

      %vec1= getWord(%vec,0);
      %vec2= getWord(%vec,1);
      %vc = %vec2;
      %vec3= getWord(%vec,2);

      %ray = %vec1;

      %vec1 = %ray*Math::cos(%rot3);
      %vec2 = %ray*Math::sin(%rot3);
      %vec = %vec1 @" "@ %vec2 @" "@ %vec3;
      %vec = Vector::add(%vec,Vector::getFromRot(%rot,%vc,0));
      return %vec;
}

function Vector::multiply(%vec1, %vec2)
{
    %vec1X = getWord(%vec1, 0);
    %vec1Y = getWord(%vec1, 1);
    %vec1Z = getWord(%vec1, 2);

    %vec2X = getWord(%vec2, 0);
    %vec2Y = getWord(%vec2, 1);
    %vec2Z = getWord(%vec2, 2);

    %vec3X = %vec1X * %vec2X;
    %vec3Y = %vec1Y * %vec2Y;
    %vec3Z = %vec1Z * %vec2Z;

    return %vec3X @ " " @ %vec3Y @ " " @ %vec3Z;
}

function Vector::divide(%vec1, %vec2)
{
    %vec1X = getWord(%vec1, 0);
    %vec1Y = getWord(%vec1, 1);
    %vec1Z = getWord(%vec1, 2);

    %vec2X = getWord(%vec2, 0);
    %vec2Y = getWord(%vec2, 1);
    %vec2Z = getWord(%vec2, 2);

    %vec3X = %vec1X / %vec2X;
    %vec3Y = %vec1Y / %vec2Y;
    %vec3Z = %vec1Z / %vec2Z;

    return %vec3X @ " " @ %vec3Y @ " " @ %vec3Z;
}

function GameBase::getMass(%obj)
{
    %mass = (GameBase::getDataName(%obj).Mass);
    if (getObjectType(%obj) == "Flyer")
        %mass += %obj.PassengerMass;
    return %mass;
}

function GameBase::getMuzzlePosition(%player)
{
        %trans = GameBase::getMuzzleTransform(%player);
        %vec6 = getWord(%trans,9);
        %vec7 = getWord(%trans,10);
        %vec8 = getWord(%trans,11);
        return %vec6@" "@%vec7@" "@%vec8;
}

function GameBase::getNormalRotation(%pos, %pos2)
{
    return Vector::Normalize(Vector::Sub(%pos, %pos2));
}
```

```
function EncryptionSystem::Encipher(%str, %key)
{
    %key = EncryptionSystem::GetKey(%key);
    %str = String::Reverse(%str);
    %slen = String::Len(%str);

    if(%slen > 64) //Too long, it'll get cut off
        return -1;

    for(%i = 0; %i < %slen; %i++)
        %sp = %sp @ " " @ String::charValueAt(%str, %i);

    %sp = String::trim(%sp);

    for(%i = 0; (%w = getWord(%sp, %i)) != -1; %i++)
        %enc = %enc @ "*" @ (%w ^ %key) << 2;

    return String::Mid(%enc, 1, String::Len(%enc));
}

function EncryptionSystem::Decipher(%str, %key)
{
    %key = EncryptionSystem::GetKey(%key);
    %str = String::Replace(%str, "*", " ");

    for(%i = 0; (%n = getWord(%str, %i)) != -1; %i++)
        %sp = %sp @ " " @ (%n >> 2 ^ %key);

    for(%i = 0; (%w = getWord(%sp, %i)) != -1; %i++)
        %final = %final @ String::fromCharCode(%w);

    return String::Reverse(%final);
}

function EncryptionSystem::GetKey(%str)
{
    %slen = String::Len(%str);

    for(%i = 0; %i < %slen; %i++)
        %n += String::charValueAt(%str, %i);

    return 2 >> %n;
}
```

```
@ECHO OFF
TITLE VOL Compile Batch
ECHO (-) Deleting Old 'Scripts.vol'...
TITLE VOL Compile Batch: Deleting Old 'Scripts.vol'
DEL scripts.vol
ECHO (-) Creating New 'Scripts.vol'...
TITLE VOL Compile Batch: Creating New 'Scripts.vol'
ECHO (-) Archiving Files...
TITLE VOL Compile Batch: Archiving Files
FOR %%f IN (*.cs) DO (
      vt -q scripts.vol %%f
      ECHO %%f
      TITLE VOL Compile Batch: Archiving %%f
)
ECHO (-) Done...
TITLE VOL Compile Batch: Done
```