ZScript v7

Programming Language

Reference Manual

Definitions

Token: A *token* is a symbol, or a string of letters, reserved by the system for internal use, that directs the parser to act in a specific manner, or is used by the parser to identify the type of data, statement, object, or script that you are using. All mathematical symbols, assign symbols, datatypes, script types, and statement types are *tokens.* For a full list of *tokens*, see the table (insert).

Scope: t/b/a

Directive: A compiler-specific instruction, that informs the compiler to do a specific task. At this time, the only legal directive is *import "filename".*

Identifier: An *identifier* is the alpha-numeric sequence of characters used to identify a variable, pointer, or function 'by name'. If you have a declaration of int x, the *identifier* is *x.*

Literal: A *literal* is a mathematical value, represented solely by numbers. 215 is a *literal*, while (214+n) where *n* is 1, is not a *literal.* (n is a *variable.*)

Variable: A *variable* is any data value that can be changed, either pre-allocated by the system (internal, global variables; and pointer variables), or declared by the user. Link->X, Game->Deaths, and a user-declared int n, are all variables. All *declarations* in ZScript are technically variables, and some may also be *pointers*.

The legal *variable* ***types*** (datatypes) are: int, float (identical to int when declared in scripts), bool, ffc, npc, item, itemdata, lweapon, and eweapon. (npcdata, and others t/b/a).

Constant (a/k/a Define): In ZScript, a constant is a value that is fixed, and once declared, cannot be altered. Constants in ZScript are effectively identical to #define in C. The purpose of declaring a constant, is to assign an identifier to a numeric value, so that it is easier to use; and to permit globally changing values assigned by constants with a single change.

Array: A series of values connected in one set, to ease iteration and access. Individual values are stored in indices (or elements) of the array. For more information, see 'Arrays' (page t/b/a).

Pointer: A pointer is a declared variable that is associated with a class, object, or array. It is always the *identifier* declared for the data in question; so in the case of ffc f, the pointer is *f*; and in the case of int array_dat[256], the pointer is *array_dat.*

When referencing a variable that belongs to a pointer. You must use the *dereference token* -> to associate it. Thus, for ffc f, if you wish to access the X variable, you would reference it as *f->X*.

The internal ZScript pointers are Game, Screen, Link, Audio, NPCData, Debug, ffc, npc, item, itemdata, lweapon, and eweapon.

Datatype: The type of data held by a variable, array, or pointer. Also the *return type* of a function. Legal datatypes are: int, bool., float, ffc, npc, item, itemdata, eweapon, and lweapon.

Declaration: A declaration is the creation of a variable, or pointer, using an *identifier.* The declaration may, or may not be *initialized*.

Example: int x;

This *declares* the *variable **x*** with a datatype of *int.*

Assign: The act of storing a value in a variable, a value in an array, or a structure in a pointer.

Examples:

X = 6;

This *assigns* the *literal value **6*** to the *variable **x**.*

float arr[3]={6,1,2};

This *declares* and *assigns* the three *indices* of the *array ***arr*** with the *literal values **6**, **1**,* and ***2**.*

arr[0] = 6;

This *assigns* the *literal **6*** to *index 0* of the array **arr.*

ffc f = Screen->LoadFFC(1);

This declares the *pointer **f*** and uses the *function LoadFFC()* using the *Screen-> pointer* to load the *structure* of the *ffc **ID 1*** into the *pointer.*

Initialise: The act of assigning an initial value to a variable, or a structure to an object pointer, or values to an array during its creation.

Example

Structure: A set of related data values, such as all the values associated with an *ffc*. These are typically loaded into a pointer, created by the user, but they may also be created by the user by creating a *struct*.

Function: A *function* is an instruction that you may call that carries out a specific set of routines. There are two primary types of functions: Internal Functions, used and reserved by the compiler, that are defined in advance; and *user-defined* functions.

Internal Functions

ZScript has a wide variety of *internal functions* that are tied to the ZScript *internal pointers* and *data structures.* These are fixed, and you may not change them, but you may define your own functions to make use of them. They are the basic building blocks of the ZScript language.

User-Defined Functions

The latter of these, are functions that you create, or that are supplied in the form of headers, or in pre-existing scripts. User-defined functions may be at either:

Global (file) scope, and thus available at any time, to any script (or other function) by calling them using their *identifier*. Global function identifiers and signatures must be unique within the context of a quest at *a file* scope, across any number of headers, or files.

Script scope functions, must have a unique identifier and signature, on a per-script basis, and thus you may define a function with the same *identifier* and *signature* in multiple scripts, that behave differently, as-needed.

Function Return Type: Functions are declared with a specific *return type*. These are generally int, float, or bool, and represent the type of data that the function *returns* when called. They may also be *complex types*, such as *ffc*, *npc*, *item*, *itemdata*, *eweapon*, and *lweapon* (t/b/a npcdata, combodata, and others).

A *return type* of void specifies a *null return.* These functions do not return a value. All other function types **must** have valid return instructions. A *return* instruction in a *void* typed function will prematurely end the function and exit its *scope*.

Function Parameters: The parameters of a function represent its inputs. A function with no parameters is legal, and would look like this:

```
void ClearArray()
{
	for ( int a = SizeOfArray(array_pointer_identifier); q <= 0; q-- )
	{
		array_pointer_identifier[q] = 0;
	}
}
```

Here, the function *return* type is void, and the function has **no input parameters.**

In contrast, this function has both a *return type* and *parameters.*

```
int GetArrayIndexValue(int ptr, int index)
{
	return ptr[index];
}
```

Function Signature: The *signature* of a function is the way it is defined, by its *return* type and by its *inputs*. The following are function declarations, and explanations of their signatures:

```
void foo(int a) { Link->X += a; }
```

Signature:

Return Type: void

Parameters: (int) -- One input, type int.

```
void foo(int a, bool b) { if ( b ) Link->X += a; else Link->X -= a; }
```

Signature:

Return Type: void

Parameters (int, bool) – Two inputs, type *int*, then type *bool*.

The ZScript parser will attempt to match *function calls* based on the *signature* of a function. If the function is *internal*, then the parser will scan for it, using a pointer if supplied with one. If it cannot find a match, it will return an error reporting a mismatch based on the function *identifier*, if there is no