AutoIt

Matthew Kennedy
CIS 343 02
Professor Woodring
December 12, 2018

Abstract

The intention of this paper is to provide an insight into the scripting language AutoIt, and its use cases in the Windows environment. Throughout the paper, the language will be looked at from several different focal points, including how data is abstracted, control abstraction, readability, writability, and reliability.

Introduction

The initial release of AutoIt was in 1999 by a man named Jonathan Bennett and his team. The language was originally intended to create automation scripts for Microsoft Windows programs, but has grown significantly since then to include other functionality (Szőcs). AutoIt is an interpreted language, meaning it has no compilation necessary, and is very lightweight. According to the official AutoIt website,

"AutoIt v3 is a freeware BASIC-like scripting language designed for automating the Windows GUI and general scripting. It uses a combination of simulated keystrokes, mouse movement and window/control manipulation in order to automate tasks in a way not possible or reliable with other languages (e.g. VBScript and SendKeys). AutoIt is also very small, self-contained and will run on all versions of Windows out-of-the-box with no annoying "runtimes" required."

AutoIt follows a procedural paradigm, similar to Python. It is executed in a "top-down" fashion, and procedures are broken down into smaller procedures. AutoIt is designed to be as small and independent as possible, as to make creating and rolling out production-ready scripts as simple as possible (autoitscript.com).

Data Abstractions

AutoIt is an interpreted language, and does not use a compiler unless creating executables. AutoIt also uses dynamic typing for variables. Because of this, AutoIt can declare an integer, as follows

$var = 0

The variable $var can be reused immediately after, this time as a string:

$var = “Hello”

Because AutoIt is dynamically typed, this is allowed. Additionally, the language uses implicit typing - a user does not need to specify what type a variable is. The interpreter will define the variable as whatever it needs to be - that is, if the variable is holding an int, then it will be an int. If it is changed to hold a String, then it is a String (autoitscript.com).

Strong/Weak Typing and Coercion

In the case of strong vs weak typing, AutoIt is an interesting case. Weak typing is where a data type can be coerced to act as another data type. For example, in C, one pointer can be used as any pointer, because they all point to addresses. Strong typing prevents this. The difference between most common languages and AutoIt is that variables in AutoIt have no members - there are no functions bound to only ints or Strings or anything else. Instead, functions mutate variables, which only hold data.

To show this, if I redeclare var

$var = 5

$var is currently dynamically typed as an int. Now, if I call the function

StringInStr($var, “Hello”)

Then there would be no issues with this, as $var would be cast to a String and checked. While this may be coercion to some, it appears to be more of just dynamic typing. In the end, ultimately, variables can be made to do pretty much anything no matter their type, so AutoIt is strongly typed (Guest).

Control Abstraction

AutoIt follows common syntax for declaring and assigning a variable.

$var1 = $var2 + $var3

In the above example, $var1 is assigned to the value of $var2 + $var3. As an interpreted language that follows a procedural paradigm, the common syntax were decided to be the best to use for the common case, as to not add any complexity in a simple language.

For precedence, the reference page lists the following as precedence, from high to low: Not, ^, */, +-, &, <> <= >= = <> ==. For example,

2 + 4 * 10

Evaluates to 42 (autoitscripts.com/docs).

Conditional Statements

AutoIt has 4 types of conditional statements:

- If-Else
- Select-Case

- Switch-Case
- Ternary

From autoitscript.com/docs, “All three statements are similar and decide which code to execute depending on the condition(s) given.”

Iteration

Iteration in AutoIt is very simple - to add to a variable, any of the common methods work, such as:

$var++, $var +=1, $var = $var + 1

Each will have the same effect on $var as the last (autoitscript.com/wiki).

Iteration also is supported in the form of loops. AutoIt has three different kinds of loops: the for loop, while loop, and the do-while loop. The for loop has the following syntax:

For $i = x to y Step z

Next

The while and do-while loop share similar syntax (do-while is inverted order of keywords), so only the while will be written:

While<expression>

WEnd

All information taken from autoitscripts.com/docs

Functions and Parameter Passing Techniques

Functions in Autoit are declared as follows:

Func foo()

EndFunc

The keyword Func (or func - the language is not case sensitive) declares the start of a function. To indicate the end of a function, the keyword EndFunc is used.

For return types, AutoIt does not specify a return - instead, a user can simply declare one if needed by declaring

return $var

Lastly, for parameter passing, the default is all parameters are passed by value. This means that a copy of the data is passed to a function instead of the actual data. However, AutoIt does support passing by reference. Below is a normal function, which has argument $var taken by value, and a second that takes $var by reference (autoitscript.com/wiki).

Func foo($var) Func foo(ByRef $var)

Scope Rules

For scoping, AutoIt supports Local, Static, and Global scoping. A local variable is scoped to the function it is declared in. The lifetime of a local variable is the same as the function's. A global variable is a variable that can be used and accessed anywhere in the program. The global variable will last as long as the program. Using the static modifier on a local variable makes it so that, while access stays to just the function, the variable will stay alive even after the function returns.

Additionally, variables are restricted to the block they are declared in - a variable declare in a for loop cannot be accessed outside of the loop (autoitscripts.com/wiki).

Package/Modules/Namespaces

AutoIt does not possess packages/modules/namespacing - however, it does possess includes, as well as include guards. The language was written in C++, so this feature was included (autoitscripts.com/docs).

Exception Handling

There is exception handling in AutoIt - although it lacks try/catch blocks, it uses what is referred to as a macro to view any error codes. The way to check this macro is by checking @error, similar to as follows:

If @error then

Do error stuff

Other than checking for errors consistently, there is no standard error handling, although users have created frameworks for this which are available online.

Note from the author -

I won't lie to you, Professor Woodring - this was the first language I ever learned. It is also a language that literally no other human I've met has ever heard of, so getting this (most likely only) opportunity to write a paper / calculator for a grade was exciting. If you're interested in something interesting in the language, I've also attached a x86 assembler I helped to write about 8 years ago. This is what got me into programming.

Thanks for reading.

References

Guest. (n.d.). AutoIt v3: Your Quick Guide - PDF Free Download. Retrieved from https://epdf.tips/autoit-v3-your-quick-guide.html

Language Reference - Datatypes. (n.d.). Retrieved from https://www.autoitscript.com/autoit3/docs/

Language Reference - Datatypes. (n.d.). Retrieved from https://www.autoitscript.com/wiki/Variables_-_using_Global,_Local,_Static_and_ByRef

Szőcs, Z. (n.d.). Automate it with AutoIt. Retrieved from https://www.todaysoftmag.com/article/2152/automate-it-with-autoit