



Using EnScript to Make Your Life Easier – Session 1

Suzanne Widup, James Habben, Bill Taroli



Session 1

Getting Started with EnScript

EnScript Basics

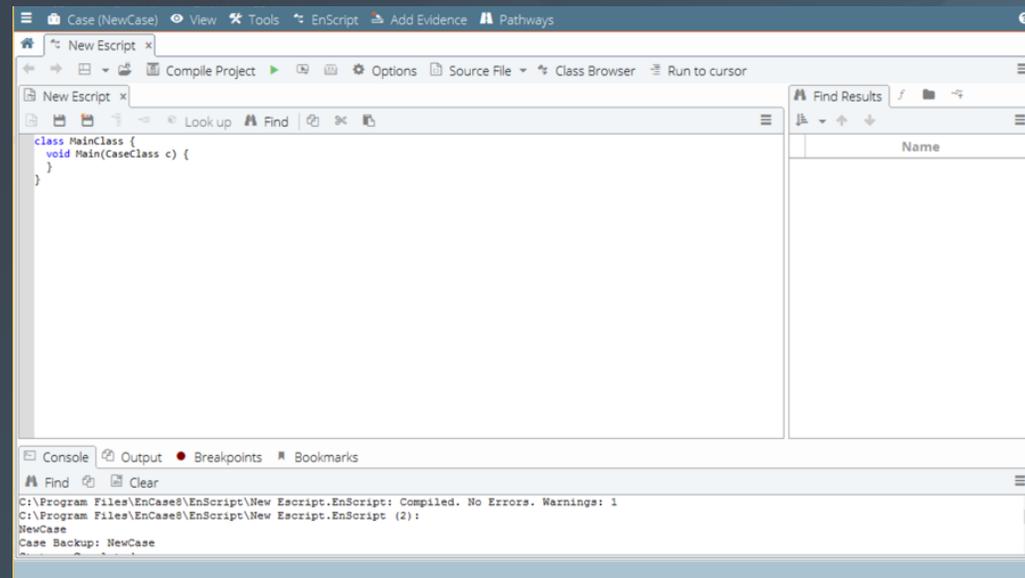
To Begin With...

- EnScript is similar to Java and C++, so if you have expertise in those languages, you'll have a head start.
- Support for COM and DCOM libraries
- You write programs in the EnScript Integrated Development Environment (IDE)
- EnScript is case sensitive, so keep that in mind when coding
- You don't need a case open to get started writing EnScripts

EnScript Integrated Development Environment

Where the magic happens

- Get Started:
 - Open EnCase if you haven't already, and choose New EnScript from the EnScript menu



The screenshot displays the EnScript IDE interface. The main editor window shows a C# class definition:

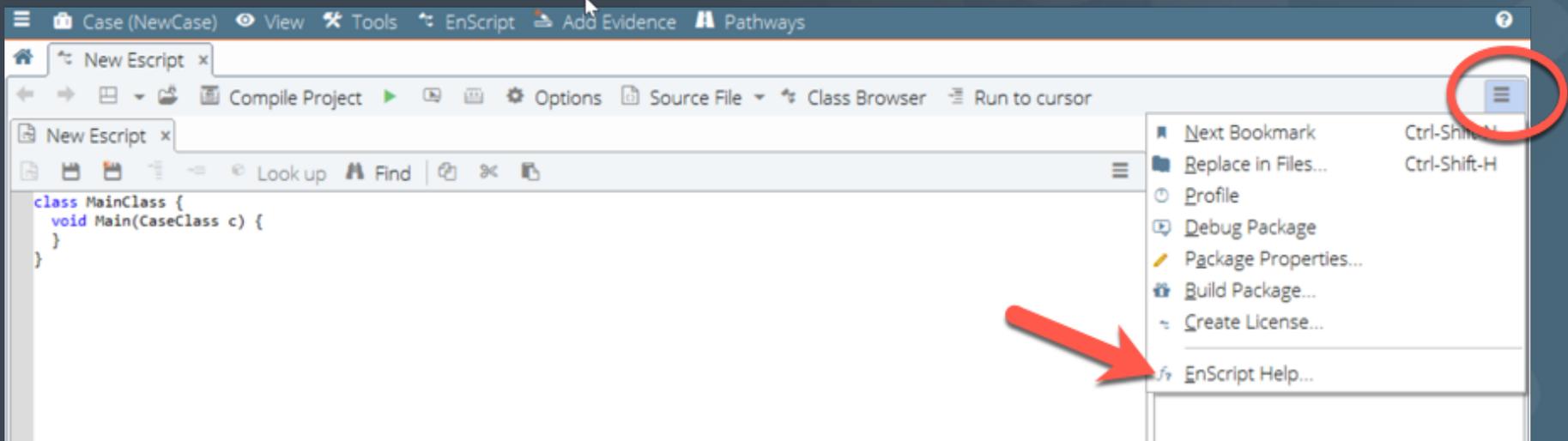
```
class MainClass {  
    void Main(CaseClass c) {  
    }  
}
```

The right-hand pane shows a 'Find Results' window with a table header 'Name'. The bottom console window displays the following output:

```
C:\Program Files\EnCase\EnScript\New EnScript.EnScript: Compiled. No Errors. Warnings: 1  
C:\Program Files\EnCase\EnScript\New EnScript.EnScript (2):  
NewCase  
Case Backup: NewCase
```

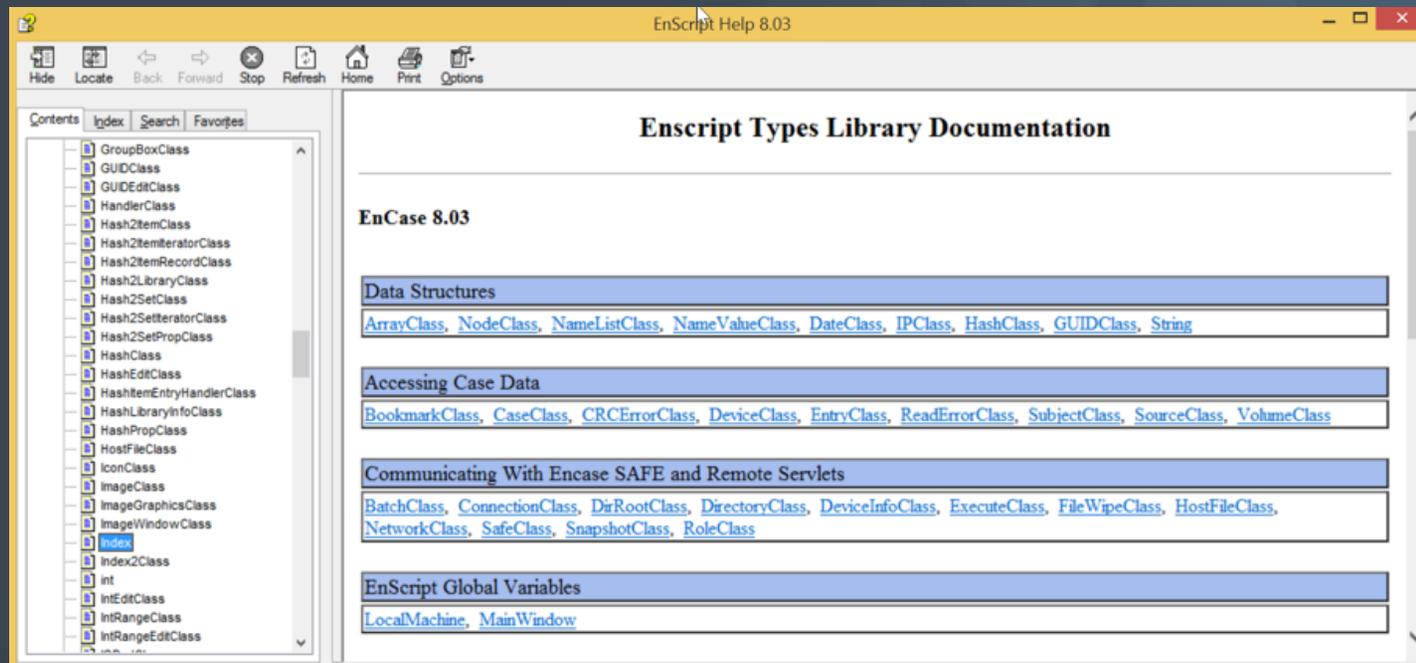
How to Learn More

EnScript Help



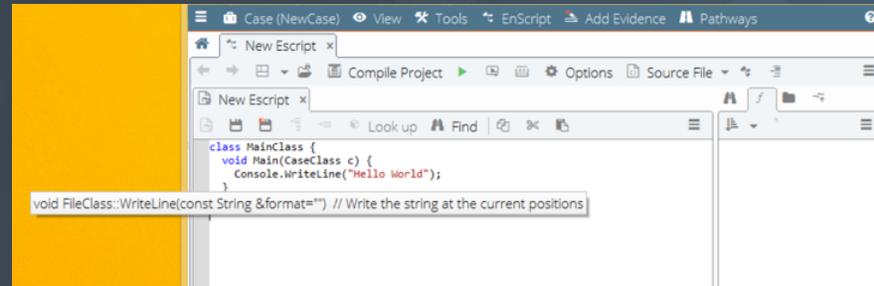
EnScript Types Library

AKA EnScript Help

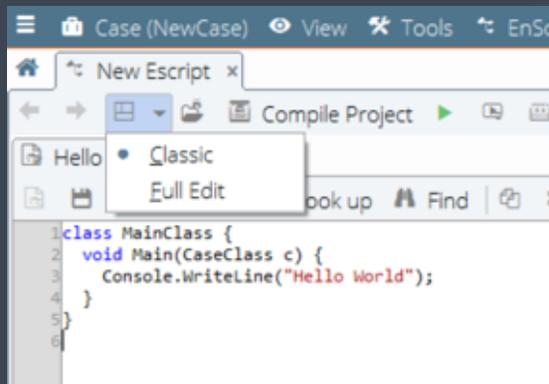


Using the IDE

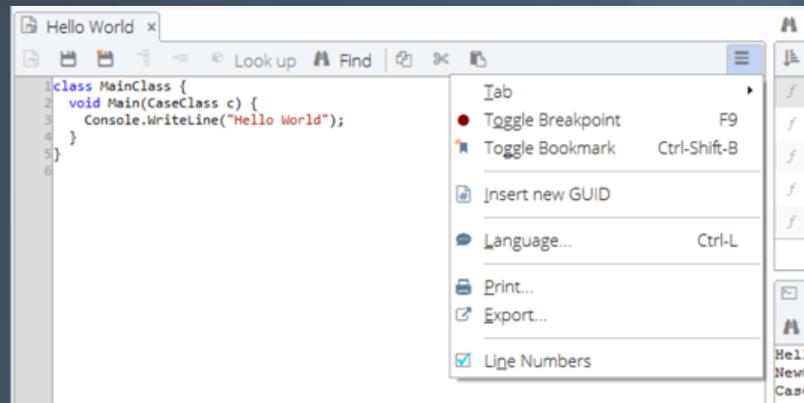
Tool Tips



Classic vs. Full Edit mode



Line Numbers



Anatomy of an EnScript

- This is what you get each time you create a new script—it is the bare minimum required for a script to run in this environment.
 - The words in blue are called reserved words. They have special meaning in the language, and can only be used as designed.

```
class MainClass {  
    void Main(CaseClass c) {  
    }  
}
```

Standard template EnScript provided with each new script as a starting point

Words in blue are EnScript reserved words

The MainClass

- Required for the code to be recognized as an executable script.
- Without the MainClass statement, the script will not stand alone—it would have to be included and called from another script that had a MainClass.
- The class definition of the MainClass is shown here and is the first and last lines only

```
class MainClass {  
    void Main(CaseClass c) {  
    }  
}
```

The Main() Function

- If the MainClass is the roadmap, these are the turn-by-turn directions.
- Just having a MainClass is not sufficient for an EnScript to run.
- The compiler will also be looking for the Main() function.
- What is a function? A function is just a chunk of code that performs a particular task.

Commenting your Code

- Comments are your friend, and it is standard practice

```
// Here is a one line comment.
```

```
/* Here is a multi-line comment. The slash asterisk combination starts the block of the comments, and to stop it, you reverse the order to end the block. It doesn't have to be on it's own line, but it is a good practice to have beginning and ending symbols line up.
```

```
*/
```

Hello World

- First program usually just writes the string “Hello World” to a console.
- You can type this in yourself in your new script, or if you run into errors (or don’t want to type in code) you can just open the ‘Hello World.EnScript’ file in our session folder for this class. That file also has the comment examples in it.
- Either way, here is what you need to have when it is ready to run:

```
class MainClass {  
    void Main(CaseClass c) {  
        Console.WriteLine("Hello World");  
    }  
}
```

- NOTE: EnScript is case sensitive. The capitalization of letters is important.

Compiling our Script

- Click on the Compile Project button. The Output tab has the results of the compile.

This compiles the project and shows any errors/warnings in the Output tab below.

```
1 class MainClass {
2     void Main(CaseClass c) {
3
4         // Here is a one line comment.
5
6         /* Here is a multi-line comment. The slash asterisk combination
7            starts the block of the comments, and to stop it, you reverse the order to end the block.
8            It doesn't have to be on it's own line, but it is a good practice to have beginning and ending symbols line up.
9            */
10
11
12         Console.WriteLine("Hello World");
13
14     }
15 }
16 }
17 }
```

This is what you want—the script compiled without syntax errors. It does not check for logic errors.

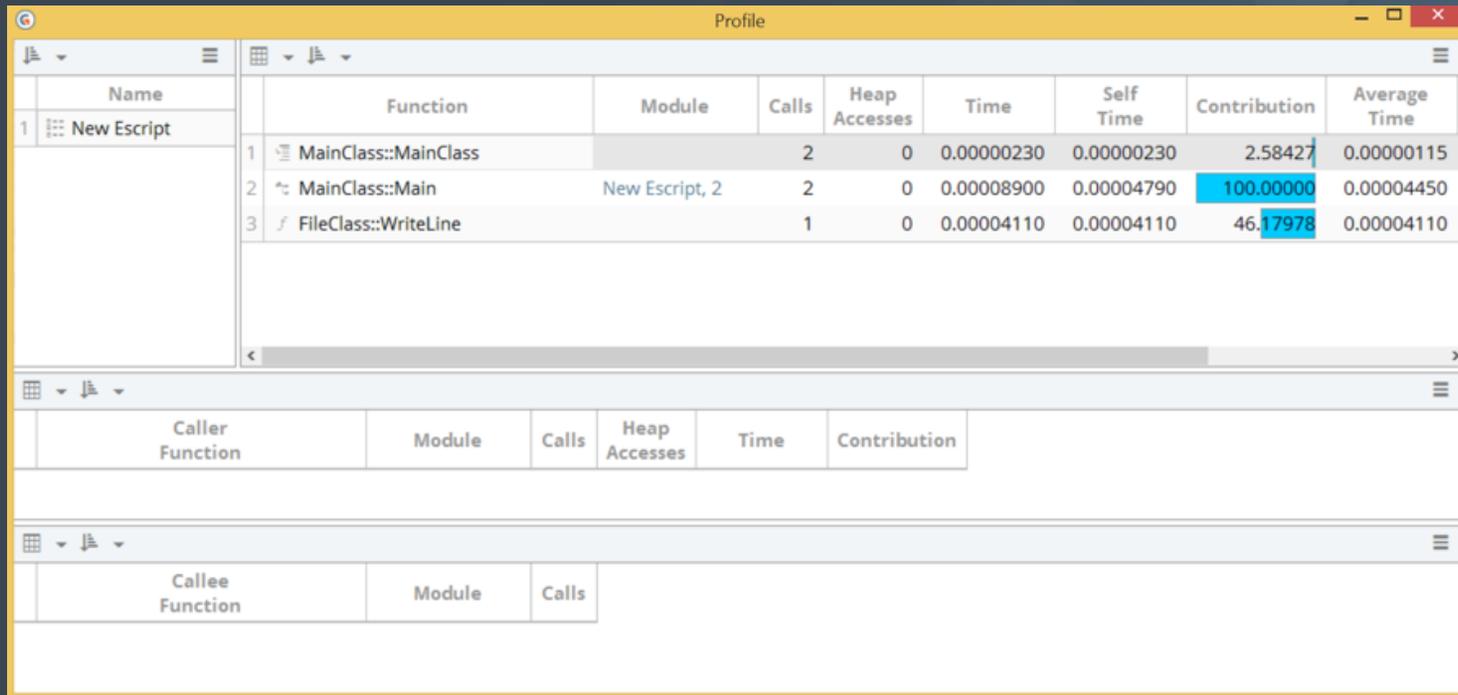
	Name
1	Compiled. No Errors. Warnings: 1, Hello World
2	Unused parameter, Hello World(2,23)

This is a warning and can be ignored. They do not prevent a script from running.

Running the Script

- Once you have the script compiling without errors, feel free to run it and see the output in the Console tab.
- You run a script using the green right facing triangle button next to the Compile Project button. 
- The Console tab is to the left of the Output tab in the bottom pane.

The Profile Browser



The screenshot shows a 'Profile' window with a table of performance data. The table has columns for Name, Function, Module, Calls, Heap Accesses, Time, Self Time, Contribution, and Average Time. The data is as follows:

Name	Function	Module	Calls	Heap Accesses	Time	Self Time	Contribution	Average Time
1 New Escript								
1	MainClass::MainClass		2	0	0.00000230	0.00000230	2.58427	0.00000115
2	MainClass::Main	New Escript, 2	2	0	0.00008900	0.00004790	100.00000	0.00004450
3	FileClass::WriteLine		1	0	0.00004110	0.00004110	46.17978	0.00004110

Below the main table, there are two smaller tables for 'Caller Function' and 'Callee Function'.

Caller Function	Module	Calls	Heap Accesses	Time	Contribution
-----------------	--------	-------	---------------	------	--------------

Callee Function	Module	Calls
-----------------	--------	-------

How does WriteLine() work?

- WriteLine() can write to other locations—anything that is a FileClass object or a type of file can be written to with this, in fact.
- Our example was writing to the Console tab—so we used the Console property (which is a reference to a FileType object) to tell EnCase what to write to.
- The dot between Console (class object) and WriteLine() (function) is how we tell the compiler we are talking about this object's properties or methods.
- Following the WriteLine(), we see the pair of parentheses and the double-quoted text we want to display within.
- We can either include text within the double quotes, or use placeholders to display the contents of variables we have used in our program.
- Finally, we have the terminating character of the semicolon, which lets the compiler know we're done with that line of code and it does not continue on a subsequent line.

What do I mean by object, property, and method?

- These are terms used in object-oriented languages. They may be new to you, so let's take a moment here and give you some definitions. Object-oriented programming languages such as EnScript refer to objects. These are data structures that have properties and methods.
- In our first EnScript, the Console is the object. A property is a characteristic of an object.
- A method, in contrast, is a function that is part of the object's class.
- Functions are simply a chunk of code that can be called by name and passed data as parameters on which to act.

Variables

- Used to store data, control program flow, and return values as parameters.
- Must be declared before use.
- Variable names cannot be EnScript reserved words, start with a number, and cannot contain characters used as logical operators in EnScript.
- Two main types:
 - Functional - natively supported in the language
 - Object – defined by a class
- When you declare a variable, it is given a default value.
 - Numbers: zero
 - Char and string: null character
 - Bool: zero
- Conversion between different variable types is handled by the compiler automatically

Controlling the Program Flow

The For Statement

- Basic syntax:

```
for (statement1; condition; statement2)
{
    Code to execute until condition is met;
}
```

The If, Else If, and Else Statements

- Basic syntax:

```
if(This > That)
{
    Console.WriteLine("This is greater than That.");
}
else if (This == That)
{
    Console.WriteLine("This and That are equal.");
}
else
{
    Console.WriteLine("That is greater than This.");
}
```

The While and Do While Statements

- Basic syntax of the while loop:

```
while(condition)
{
    Code to execute;
}
```

- Basic syntax of the do while loop:

```
do
{
    Code to execute;
}
while(condition);
```

The Break and Continue Statements

- As with other languages, the break statement will allow you to force an exit from a logical loop. The continue statement will stop this iteration of the loop that is executing and continue on with the next. These are frequently used as boundary statements where minimum and maximum values are enforced, but there are other uses.

Here is the syntax inside of an if statement:

```
if (true)
{
    continue;

    /* This is met, so it stops this loop's execution and moves back up
       to the if statement to test the condition again. */
}
else
{
    break; //this breaks us out of the loop entirely.
}
```

- Infinite logic loop—don't try this at home.

The Foreach, Forall and Forroot Statements

- Basic syntax:

```
String Path = "c:\\Data\\Myfiles";  
foreach (char element in Path)  
{  
    Console.WriteLine("{0}", element);  
}
```

Functions

- Blocks of code that perform a specific job.
- You then call them from the main function to invoke them.
- Why have functions?
 - Code reuse is a good thing—why solve a problem more than once?
 - Breaking code into chunks makes it more manageable
 - If you have to make a change to the function, you only have to do it in one place
 - If you have multiple people writing EnScripts, you can share functions that perform useful jobs between scripts.

Classes

- Everything you create in EnScript is created inside a class—whether it is the MainClass() or some other class, you are using them when you program in this language.
- A class is basically the blueprint or specification that you use to instantiate (create a specific instance of) individual objects.

The AirCRAFT Class

- A class can be declared inside the mainClass before or after it.
- You want to define associated properties, which you can see in our code reference
- The constructor is a special type of function, with the same name as the class. It serves as an initializer for the class. Not all classes require a constructor – how to tell? The Matching Symbols tab.
 - The script must be compiled and run at least once.
 - Highlight the name of the class and press the F1 key (or right click and choose Lookup). The Matching Symbols tab will show the class name.
 - Double click the class name in the Matching Symbols tab and the window will populate. See the wrench icon next to the first line? That indicates the class needs a constructor. Some classes have multiple constructors, so this is how you can see what they are and which one to use.
- If you forget your constructor, the script will compile as expected without errors. But when you run the script, you get an error “Reference to null object”.

Common EnScript Classes

- EnCase has LOTS of classes
- Classes that let you access case data (* means we will talk about it in session 2):
 - BookmarkClass
 - CaseClass
 - DeviceClass
 - EntryFileClass *
 - EntryClass *
 - ItemIteratorClass *

EnScript Resources

- <http://www.encasebook.com> is the site of Suzanne's book. The code examples from the book are all available for download from the site, and this deck and the files from this class will also be available there.
- <http://www.forensickb.com>: He is currently moving his EnScripts to another location, but when he finishes this will be the place to look to find out where they are now.
- Jon Stewart's blog: <http://codeslack.blogspot.com/search/label/EnScript>
- Guidance Software's blog: <http://encase-forensic-blog.guidancesoftware.com/search/label/EnScript>
- Guidance Software EnScript Documentation:
<http://download.guidancesoftware.com/c25XatJ%2BMLtpof3YVpmEOIB3T0iA31eTM%2BEhCiZn3qasSw8q8H6/pCBa%2BDoCHVte>

(downloads the SDK as a zip file)

Thank You

Suzanne Widup | Verizon | suzanne.widup@verizon.com | @SuzanneWidup
James Habben | Verizon | james.habben@verizon.com | @JamesHabben
Bill Taroli | Independent Contractor | bill.taroli@billsden.org | @btaroli