Article
Sean Connelly · Apr 13, 2017 3m read

Compilation gotchas and a request for change

On the back of my recent post on writing bug-less code I wanted to raise a few suggestions (to ISC) that would help prevent certain types of bugs at compile time. I've probably missed a few, but these are the main ones in my mind. Please contribute more suggestions.

Btw, these also serve as potential gotchas for new COS developers.

I except that introducing these types of changes can cause legacy code problems. Particularly where developers do some interesting overloading of method arguments. Therefore I would see these as an option that would need to be enabled and that it won't prevent compilation (it would warn not error).

As an alternative I could see these being implemented as a Lint tool. This is something I (and perhaps the community) would be willing to do if we had an open source AST tree as a starting block (if anyone is aware of one in existence for COS).

Apologies if many of these have been raised before, but it never does any harm to keep raising them. They might seem like small issues, but the accumulative time saved by the community in tracking down these bugs will be in the thousands of man days.

1. Warn on potential <COMMAND> errors

If a method has a return type then the compiler should warn of any quit code that does not return a value.

```
ClassMethod HelloWorld() As %String
{
  if 1 quit // <- this will cause a command error
    quit "Hello World"
}
```

(Its interesting that studio will complain when you try and quit a for loop with a value, but it wont complain when you try and quit with nothing on the method.)

The same problem happens in reverse, if a method does not return a value then the compiler should warn if the calling code is trying to use a return value

```
ClassMethod HelloWorld()
{
  quit //with nothing
}
```

Elsewhere...

```
set foovar=##class(Example).HelloWorld() // <- the compiler should warn about this, as it will cause a command error
```

2. Warn on incorrect return type

If a method has a return type (e.g. As %String) then the compiler should warn of any quit code that returns the wrong type.

3. Warn on incorrect argument type

Arguments in a method can have a declared type. Unfortunately, these type declarations are not enforced in any way. If the calling code passes in the wrong type then it can cause a) crashes, b) worse, silent problems.

```
ClassMethod HelloWorld(pObject As Foo.Bar) As %String
{
   Write !, pObject.foo // this will fall over if passed a string
}
```

Elsewhere...

```
Do ##class(Example).HelloWorld("Hello") // this should cause a compiler warning
```

4. Warn on incorrect number of parameters

Calling code should supply the correct number of parameters unless the Formal Spec of the arguments include a Variadic argument, e.g.

```
ClassMethod HelloWorld(...) As %String
```

However, this would also require some kind of notation that would allow arguments to be optional, for instance TypeScript fixes this problem in JavaScript by implementing a question mark after the argument.

```
ClassMethod HelloWorld(pRequired As %String, pAlsoRequired As %String, pOptional? As %String) As %String
```

intializers and optional's would not be allowed to work together.

```
ClassMethod HelloWorld(pRequired As %String, pAlsoRequired As %String, pOptionalBad? As %String="") As %String
```

As an alternative, just an empty intializer could be used to mark an argument as optional

5. Enable auto suggestion for methods that return objects as an output of the arguments.

```
Do ##class(Foo.Bar).DoThing(pArg1, .pArg2)
```

Currently if pArg2 is an object then we have to add a #dim pArg2 to the code to get the auto suggest to work

6. Warn on properties and methods that are not members of the object.

Currently the compiler will error on trying to access a non existent property or method of this class, e.g.

```
Do ..ThisDoesNotExist() //causes compilation error
```

But if I do the same on an object instance, then I will not get any type of warning...

```
ClassMethod HelloWorld(pObject As Foo.Bar) As %String
{
  Do pObject.ThisDoesNotExist() //wont fail until run time
}
```

#Compiler #ObjectScript #Caché