

Article

[Laurel James \(GJS\)](#) · Apr 20, 2021



3m read

[Open Exchange](#)

## Why gj :: locate?

You may think it isn't too difficult to get from label+offset^routine to the actual source line responsible for the error. For an expert it isn't that hard... most of the time. But there are enough oddities and special rules that even an expert can get it wrong, whilst spending a lot of time trying to get there.

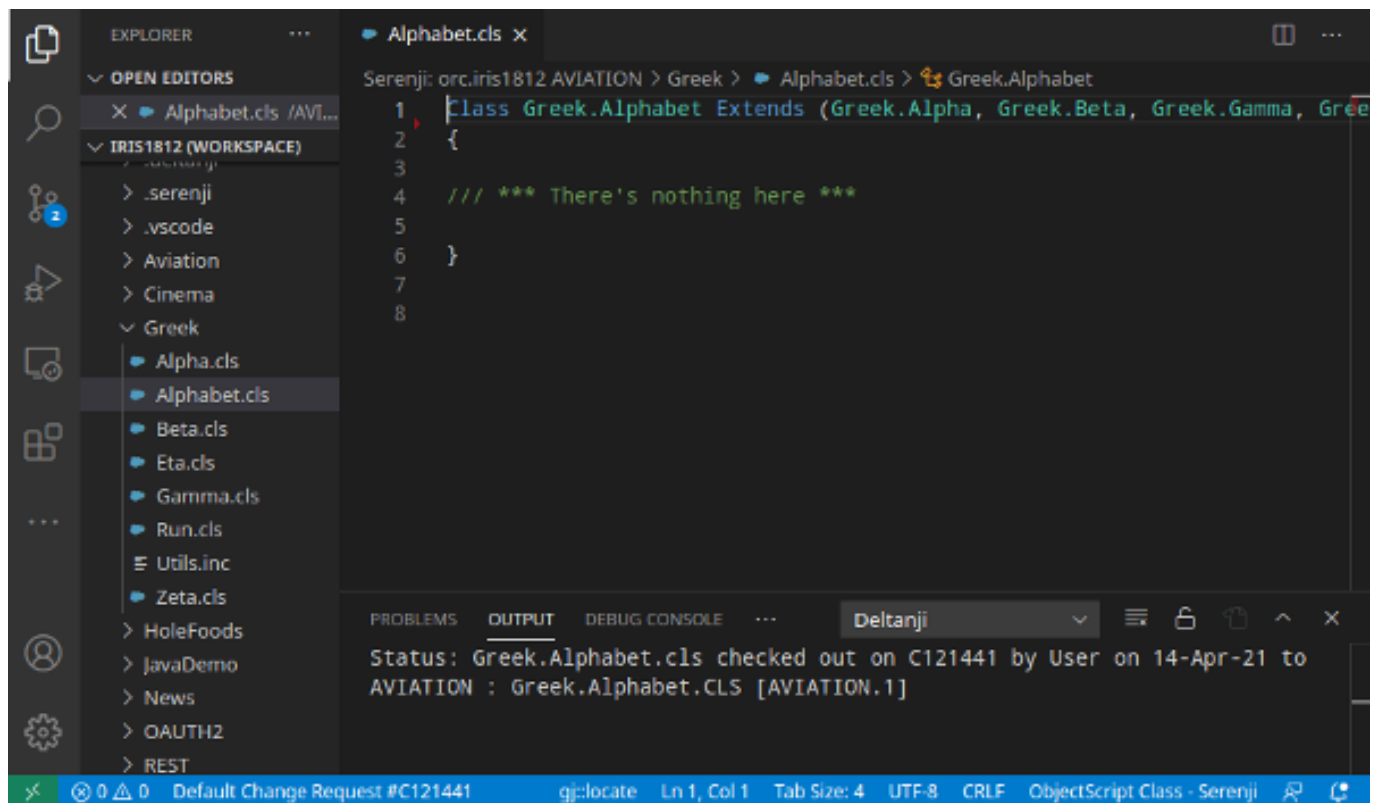
**gj :: locate** is the latest tool from George James Software – it debugs any error, class or routine by converting the location of an error in compiled .int code to the corresponding location in your source, and then taking you right there.

Image this scenario... You've got a deadline looming and a mission critical problem in your Greek Alphabet application has been identified with this error code:

```
set characterCount = characterCount + 1
<UNDEFINED>zeta+5^Greek.Alphabet.1 *characterCount
```

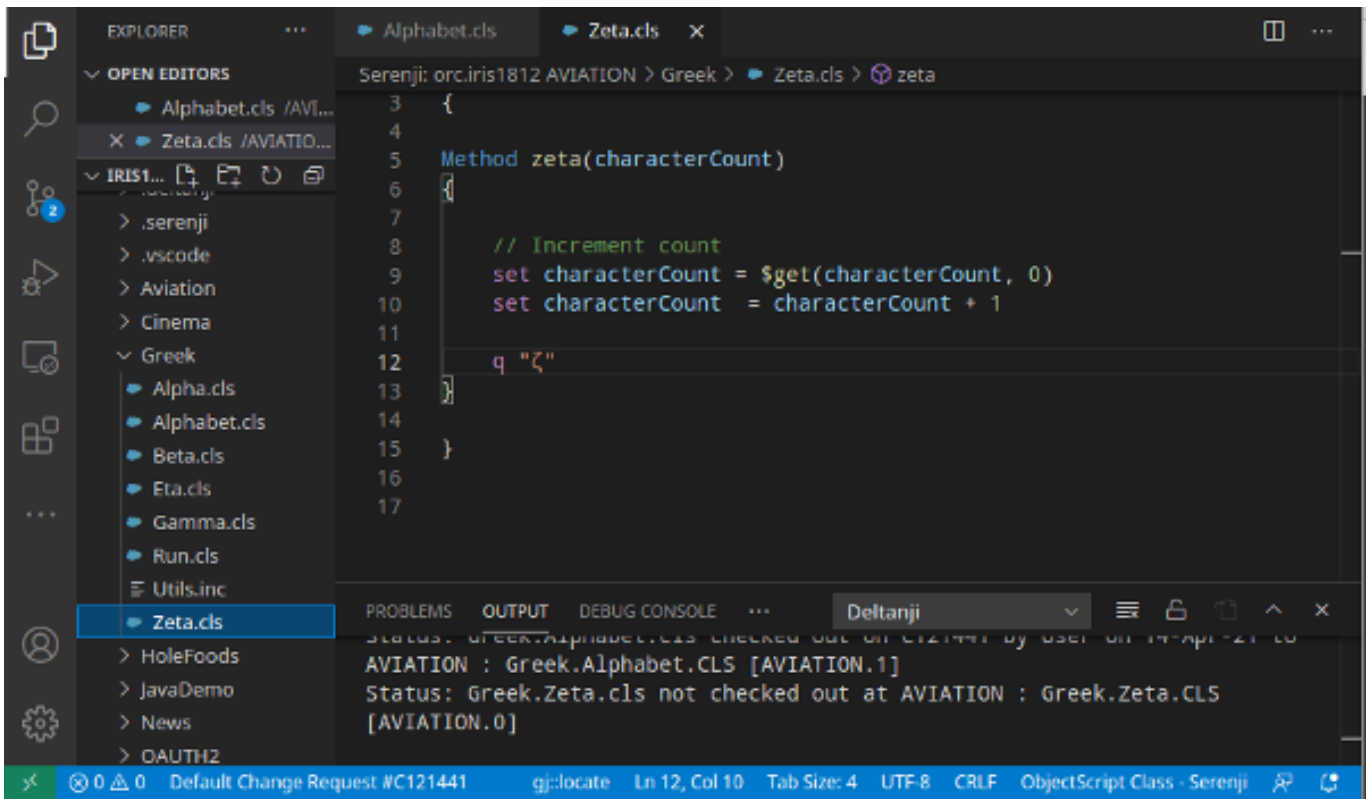
How do you find the corresponding source code line for this error? There are two ways you could approach it -

For the first, you have to know a few things. Like, you have to know that when a routine ends with a period followed by a number it means it's a class, not a routine. So now you go open up Greek.Alphabet.cls.



Yes, the error message says the problem is in the Greek.Alphabet class, but as you can see, it's empty.

Closer inspection reveals that the source code must have come from one of the superclasses. Greek.Zeta.cls would be a good guess. Let's look in there.



Well, here we are. It's obviously line 10 which is `zeta+5`. But hang on, the error message was `<UNDEFINED>` and the undefined variable is `characterCount`. How can that possibly be? `characterCount` is clearly being set on the previous line. This error is impossible!

Now you're feeling a bit challenged, you're spending longer than you'd like on this – and that deadline getting closer and closer.

Then you remember! Methods are given a `z` prefix when compiled into an `.int` routine. Aha! Zeta is the 6th letter of the Greek alphabet, but the 7th letter is Eta. We should be looking for a method called `eta`.

Let's look at `Greek.Eta.cls`

```
1  /// gj::locate example
2  Class Greek.Eta
3  {
4
5  Method eta(characterCount)
6
7
8      #include Greek.Utils
9      set characterCount = $get(characterCount, 0)
10     set characterCount = characterCount + 1
11
12     quit $$$eta
13 }
14
15
16
17
```

PROBLEMS OUTPUT DEBUG CONSOLE ... Deltanji  
Status: Greek.Zeta.cls not checked out at AVIATION : Greek.Zeta.CLS [AVIATION.0]  
Status: Greek.Eta.cls not checked out at AVIATION : Greek.Eta.CLS [AVIATION.0]

Hmm. It's nearly identical to Greek.Zeta.cls and characterCount has clearly been initialised on the previous line. What's going on?

More head scratching, and more checking the clock. Perhaps the problem is something to do with that #include line. Let's look in there.

```
1  // Character Utility Macros
2
3  // Increment Counter
4  set characterCount = $get(characterCount, 0)
5  set characterCount = characterCount + 1
6  set status = "pending"
7
8  #define alpha "α"
9  #define beta "β"
10 #define gamma "γ"
11 #define delta "δ"
12 #define epsilon "ε"
13 #define zeta "ζ"
14 #define eta "η"
15 #define theta "θ"
16 #define iota "ι"
17 #define kappa "κ"
```

PROBLEMS OUTPUT DEBUG CONSOLE ... Deltanji  
Status: Greek.Eta.cls not checked out at AVIATION : Greek.Eta.CLS [AVIATION.0]  
Status: Greek.Utils.inc not checked out at AVIATION : Greek.Utils.INC [AVIATION.0]

Line 5 looks promising. But... oh no, characterCount can't be undefined here. It's impossible.

So, eventually, after scrolling down 20 or so lines you find some more code:

```
14 #define eta "η"
15 #define theta "θ"
16 #define iota "ι"
17 #define kappa "κ"
18 #define pi "π"
19 #define chi "χ"
20 #define psi "ψ"
21 #define omega "ω"
22
23 // Increment counter
24 kill characterCount
25 set characterCount = characterCount + 1
26
27
28
29
30
```

PROBLEMS OUTPUT DEBUG CONSOLE ... Deltanji

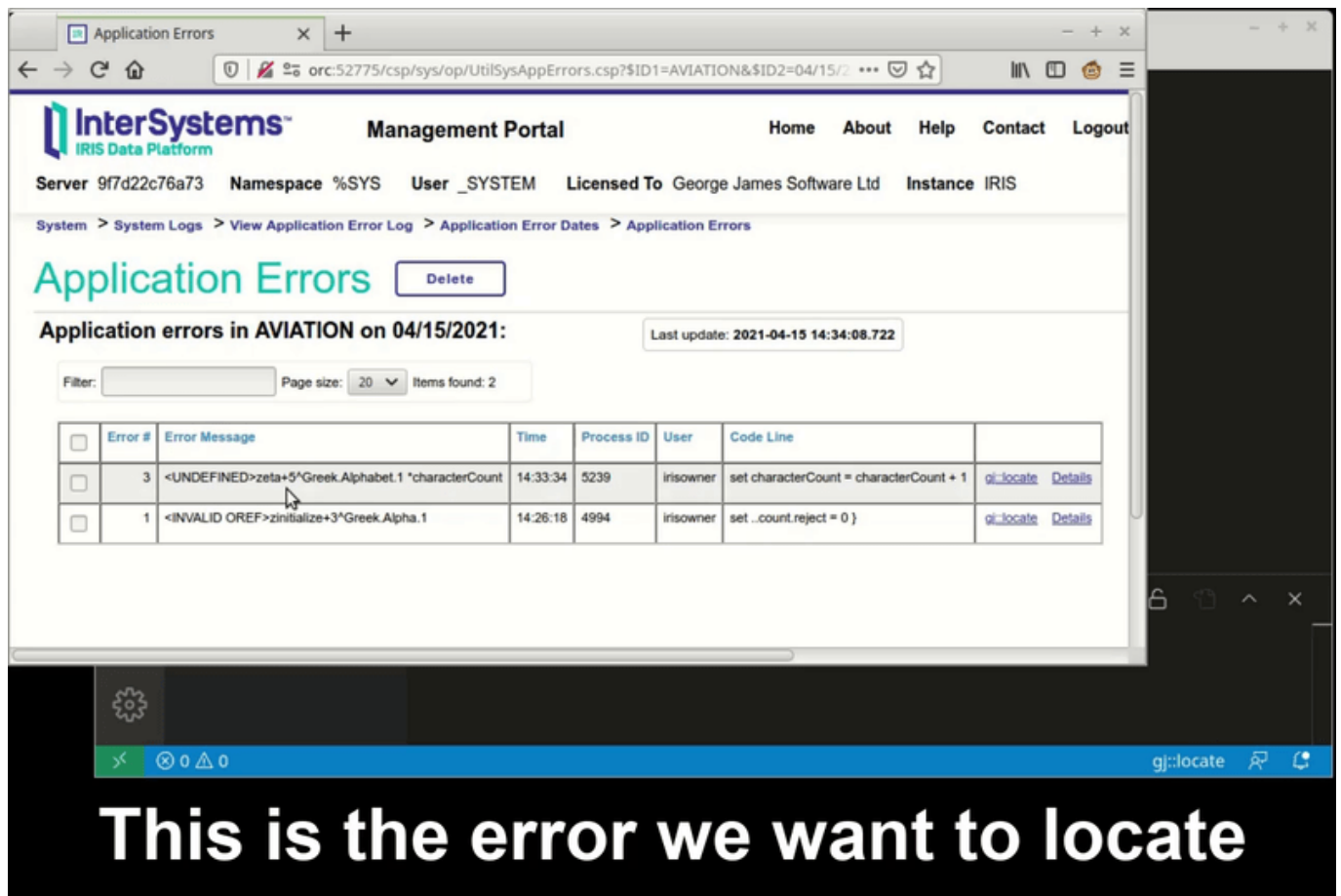
Status: Greek.Utils.inc not checked out at AVIATION : Greek.Utils.INC [AVIATION.0]

Status: Greek.Utils.inc not checked out at AVIATION : Greek.Utils.INC [AVIATION.0]

gj::locate Ln 1, Col 1 Tab Size: 4 UTF-8 CRLF ObjectScript Include - Serenji

Now we know why characterCount is undefined, and fortunately it looks like an easy fix. The real error is at line 25 of Greek.Utils.inc. Phew, finally got there.

Remember that second way we mentioned you could approach this? Well, here's how to quickly find the error using **gj::locate** -



... and you've even got time to make a cuppa before you get back to what you were working on before.

So, this is a somewhat contrived example that spotlights some of the many gotchas you might encounter, however getting from an error message to the corresponding source line can be difficult, especially when you're working to a pressing deadline.

**gj::locate** is George James Software's entry to the current developer tools contest. Try it out [here](#), and if you'd like to vote for us, click [here](#).

[#Contest](#) [#Debugging](#) [#Development Environment](#) [#Management Portal](#) [#ObjectScript](#) [#VSCode](#) [#Caché](#)  
[#InterSystems IRIS](#) [#InterSystems IRIS for Health](#) [#Open Exchange](#)  
[Check the related application on InterSystems Open Exchange](#)

Source URL: <https://community.intersystems.com/post/why-gj-locate>