Article
Yuri Marx · Sep 26, 2022 9m read

Debug the ObjectScript code using VSCode

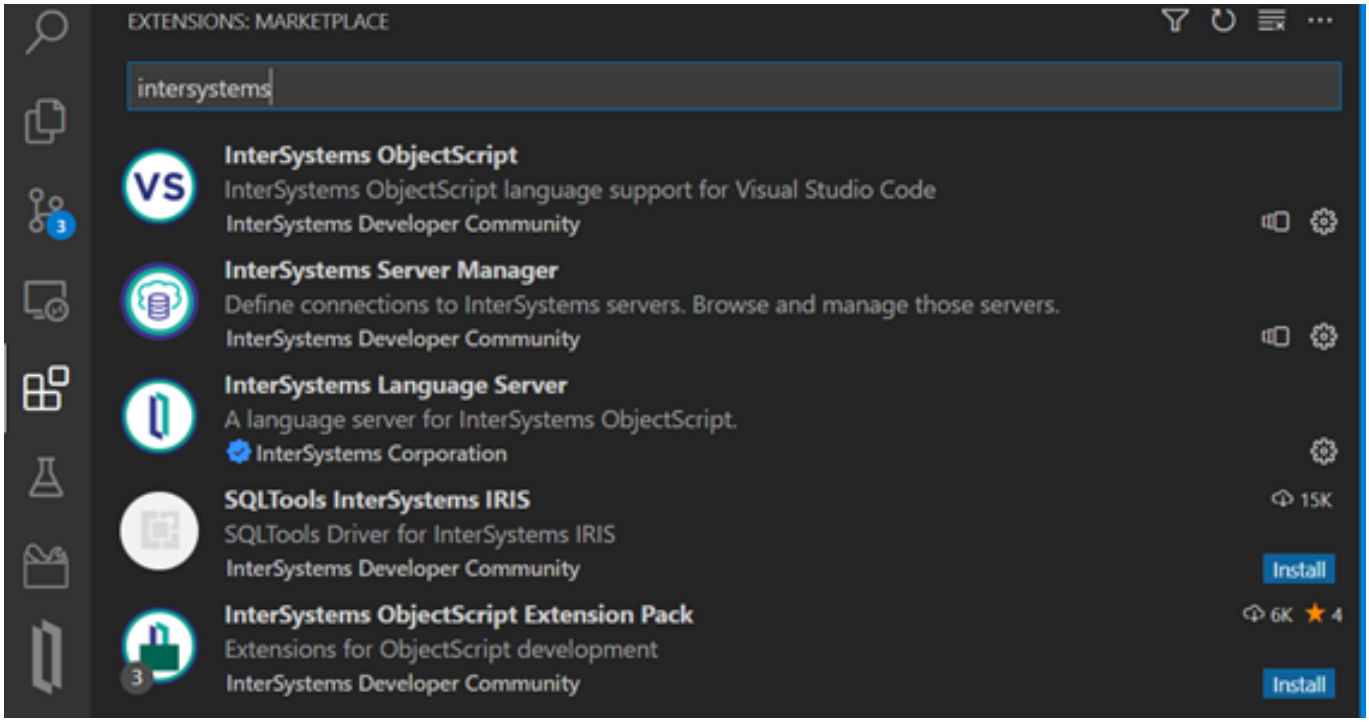
Visual Studio Code (VSCode) is the most popular code editor on the market. It was created by Microsoft and distributed as a free IDE. VSCode supports dozens of programming languages, including ObjectScript, Until 2018, Atelier (based on Eclipse). It was considered as one of the main options to develop InterSystems products. However, in December 2018, when the InterSystems Developer Community launched support for VSCode, a relevant portion of InterSystems professionals started actually using this editor and have been doing it ever since, especially the developers working with new technologies (Docker, Kubernetes, NodeJS, Angular, React, DevOps, Gitlab etc.). Some of the best features of VSCode are the Debug capabilities. That is why this article will demonstrate in detail how to debug ObjectScript code, including class code and %CSP.REST code.

What is debugging?

Debugging is a process of detecting and resolving “ bugs ” - errors in your ObjectScript code.Some errors are logical, not compilation ones. That is why to understand them in your source code execution logic, you need to see the program running line by line. This is the best way to identify the conditions or programming logic and find logical mistakes. Other errors are run-time ones. For us to be able to debug this kind of issue, it is essential to check the variable values assigned first.

Install the InterSystems IRIS extensions for VSCode

First of all, you must install InterSystems IRIS extensions into your VSCode IDE. To do it, go to Extensions, look for InterSystems and install these InterSystems extensions:



The last extension (InterSystems ObjectScript Extension Pack) should be installed first, because it is a pack of required extensions to connect, edit, develop and debug ObjectScript. The other extensions listed are optional.

Sample application

We will use an application from my GitHub repository (https://github.com/yurimarx/debug-objectscript.git) for the debug examples in this article. So, follow the steps below to get, run and debug the application in VSCode:

1. Create a local directory and get the project source code into this directory:

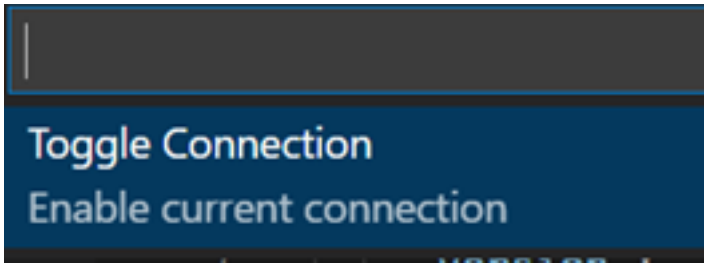
```
$ git clone https://github.com/yurimarx/debug-objectscript.git
```

2. Open the terminal in this directory (iris-rest-api-template directory) and run:

```
$ docker-compose up -d --build
```

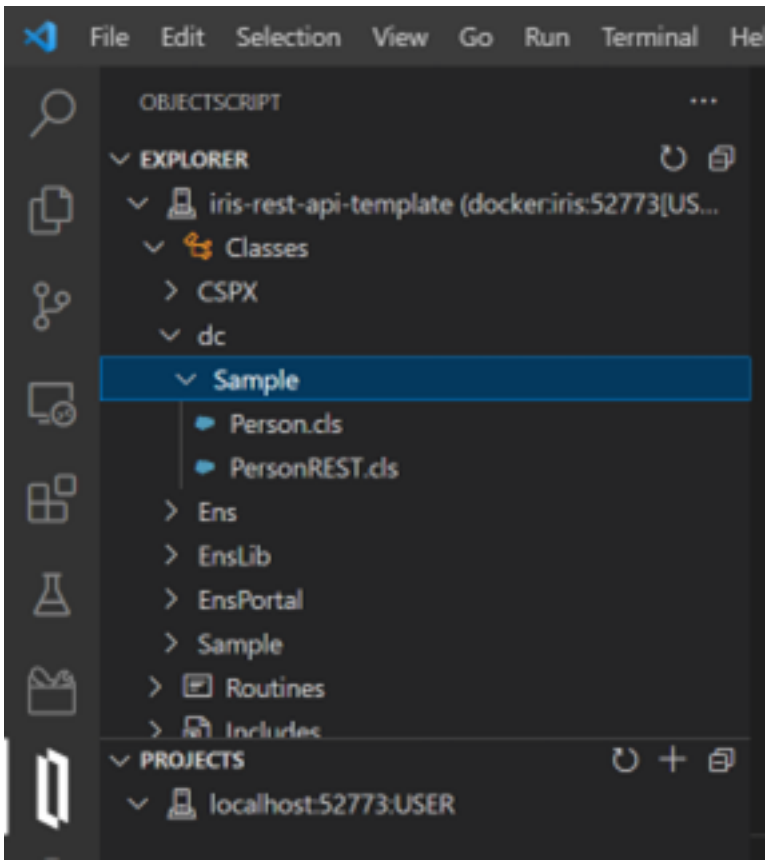
3. Open VSCode inside the iris-rest-api-template directory using $code . or click-right on the folder and select Open with Code.
4. In the footer, click ObjectScript to open the options configured in .vscode/launch.json:

5. Select Toggle Connection to enable the current connection:

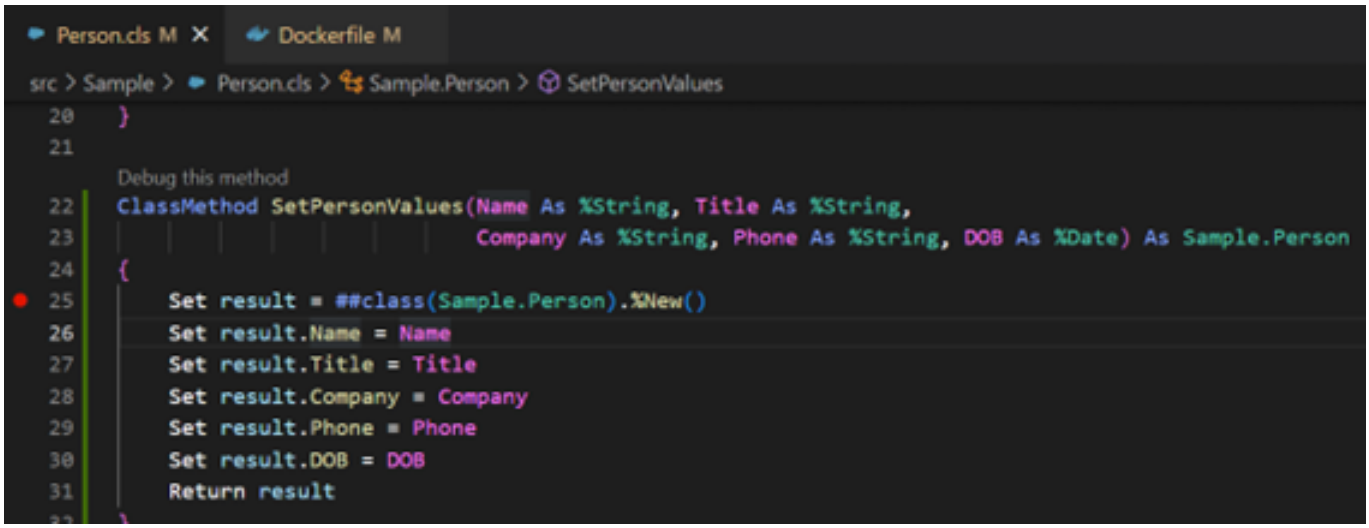


6. After enabling this connection, you will get the connection shown below:

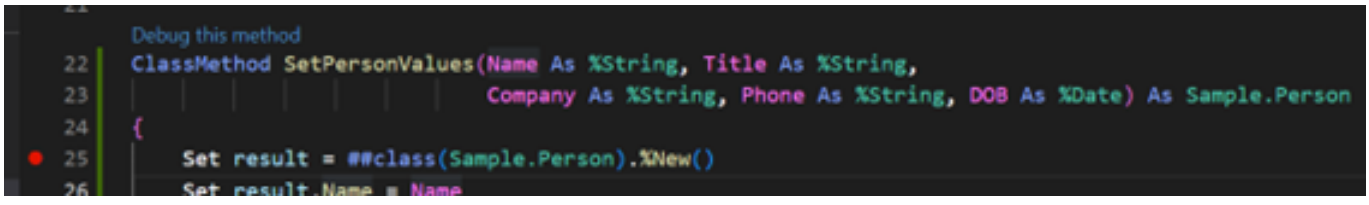
7. If you are not connected with IRIS in VSCode, you will not be able to debug the ObjectScript code. So check your connection by clicking on the ObjectScript VSCode extension:



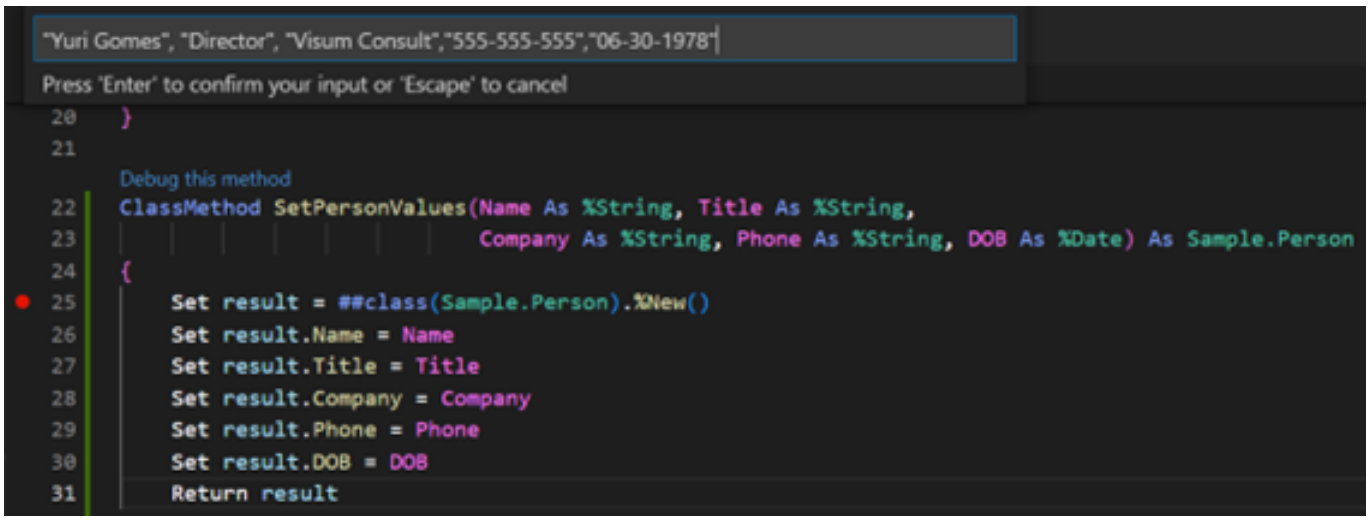
8. Open Person.cls, go to line number 25, and point the mouse to the number 25 on your left.



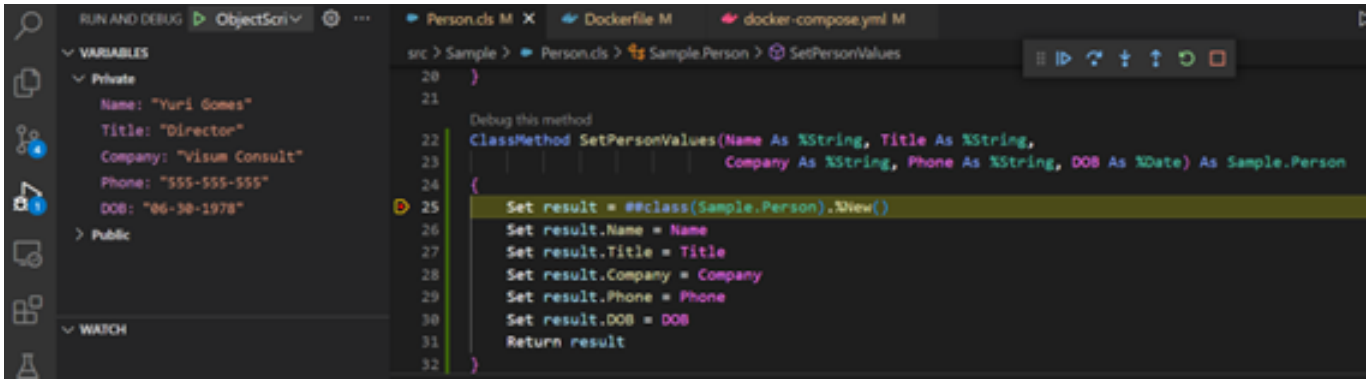
9. Now, point mouse on top ClassMethod and click on Debug this method:



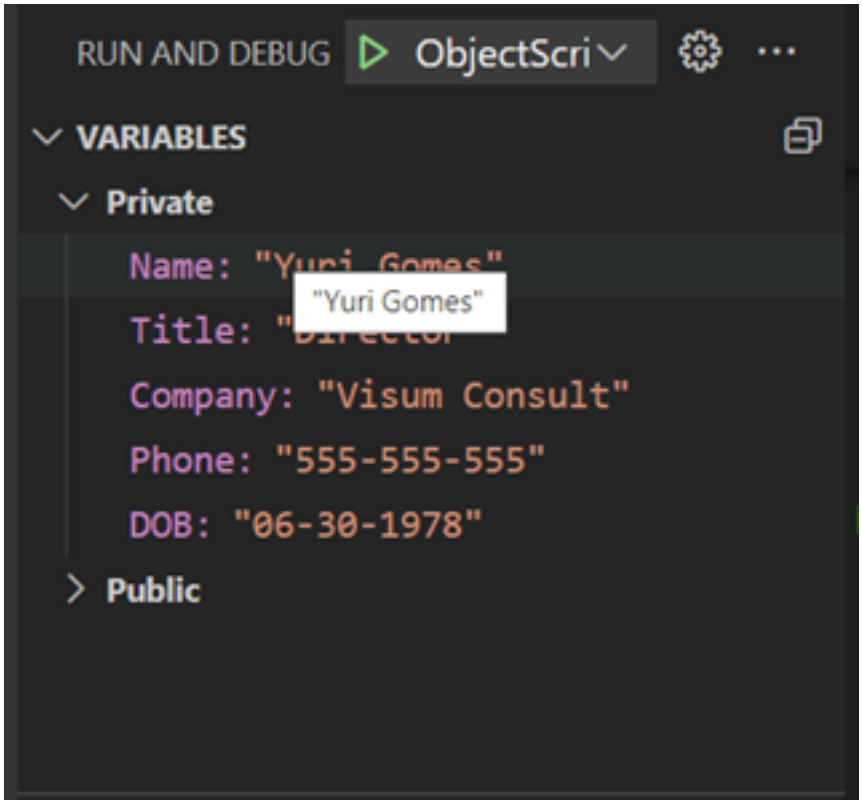
10. VSCode will open a Dialog for you to set values for the parameters, so set the values as demonstrated beneath, and click Enter:



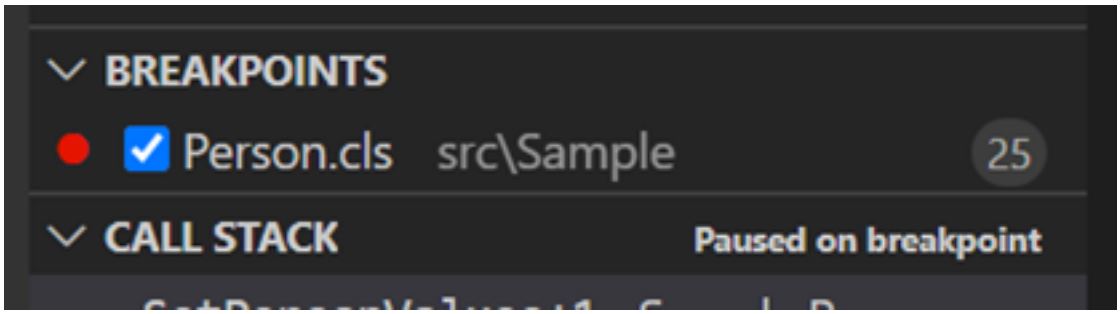
11. VSCode will break the code execution on line 25:



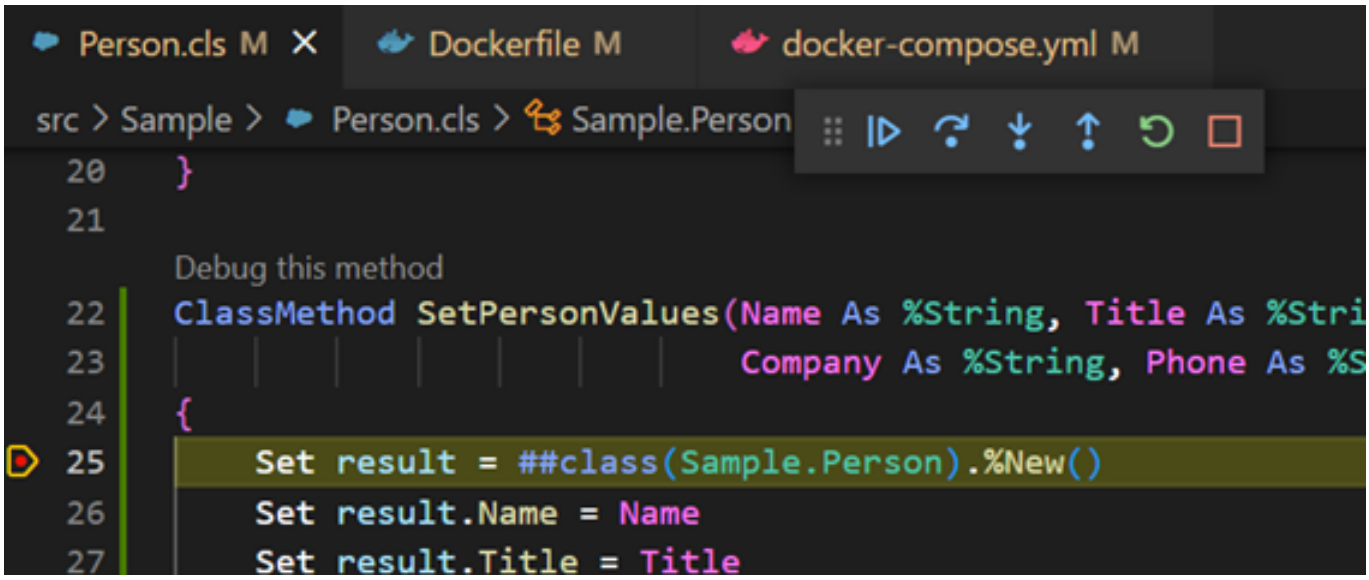
12. Now, on the top left, you can see current variable values.



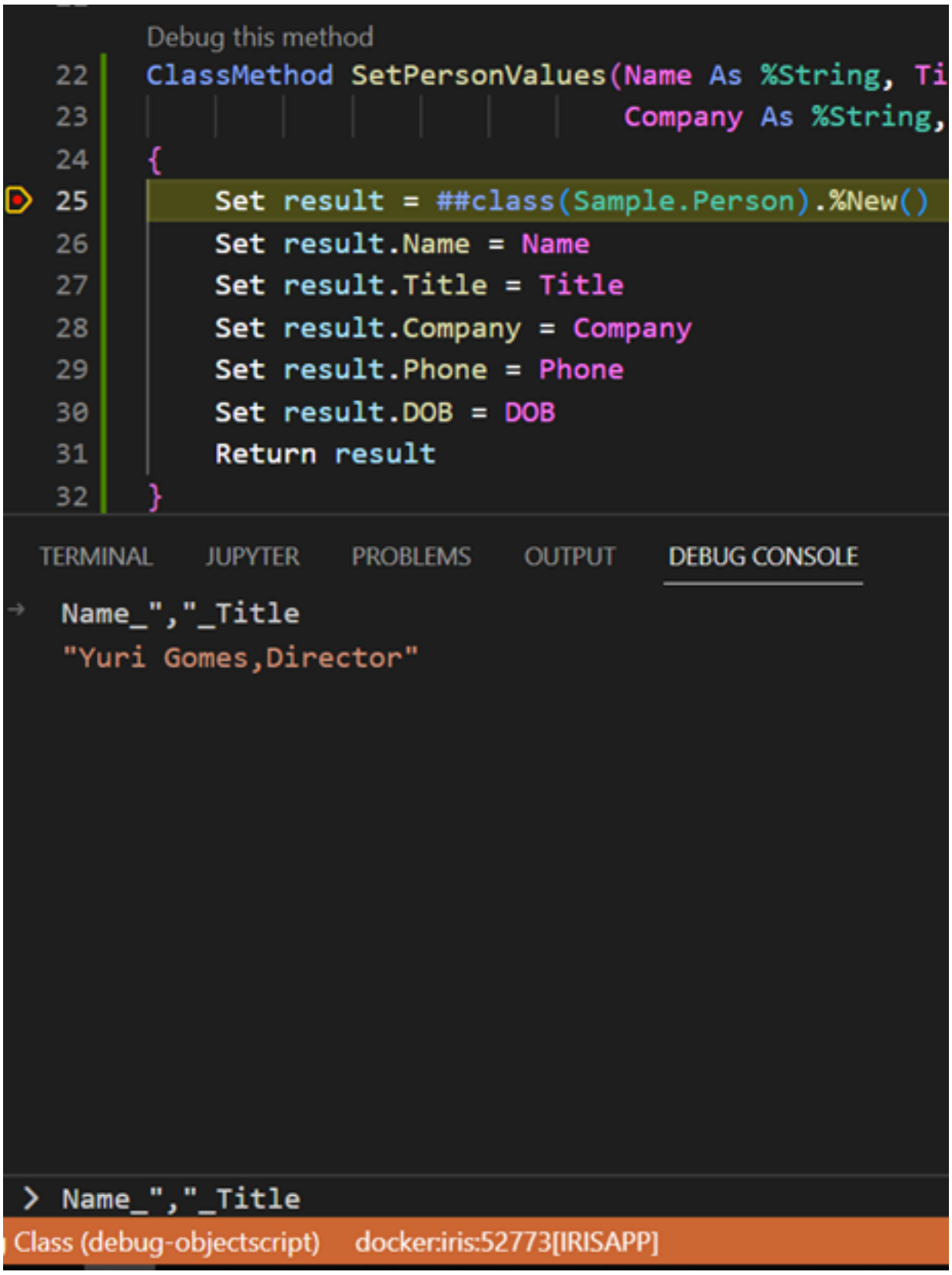
13. On the bottom left you can now see the breakpoint list and the call stack for the current breakpoint.



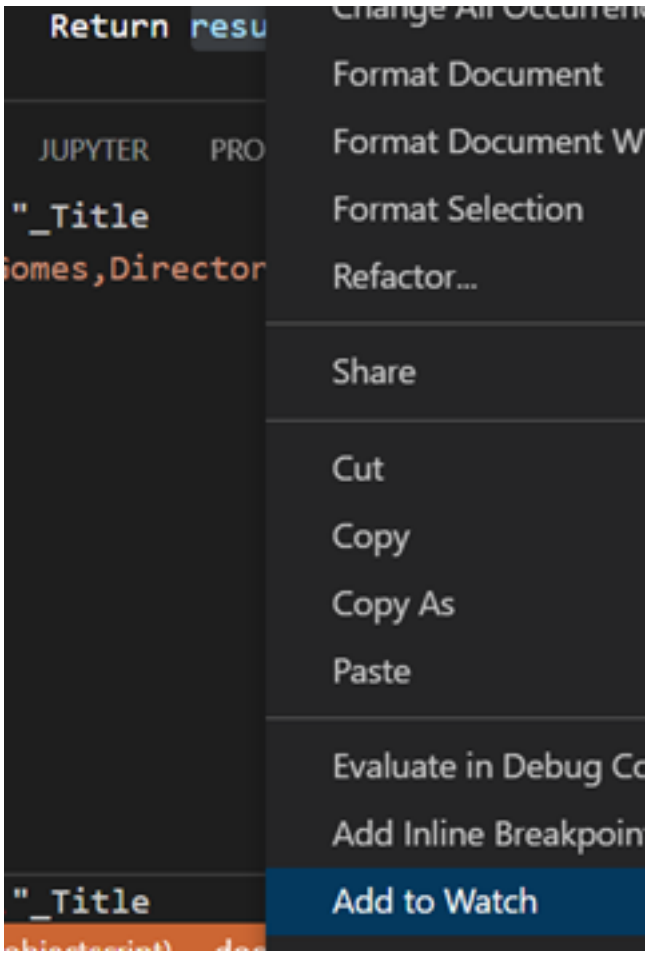
14. On the top right you can see the source code with the breakpoint and a set of buttons to control the execution of debugging.



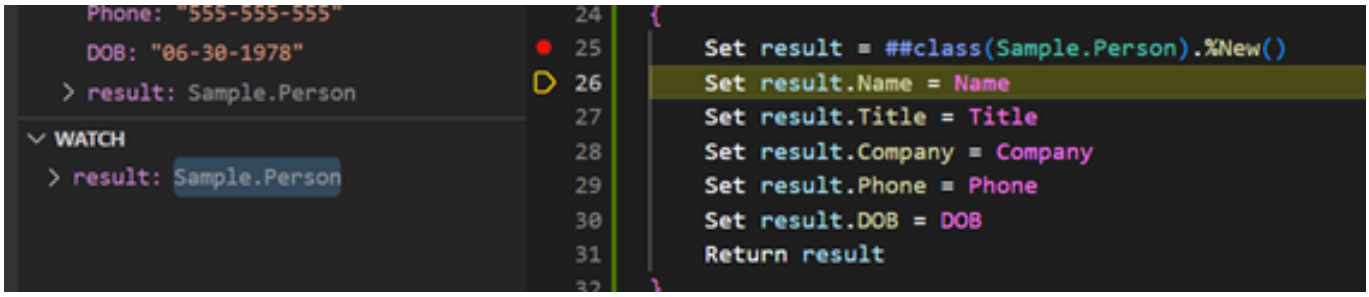
15. If you need to use Debug Console to type something there, type: Name_" , " Title to see variables concatenation using Debug Console:



16. Select the result variable, click the right mouse button and Select Add to Watch:



17. Now you can monitor or change the current value of the result variable in the WATCH section:

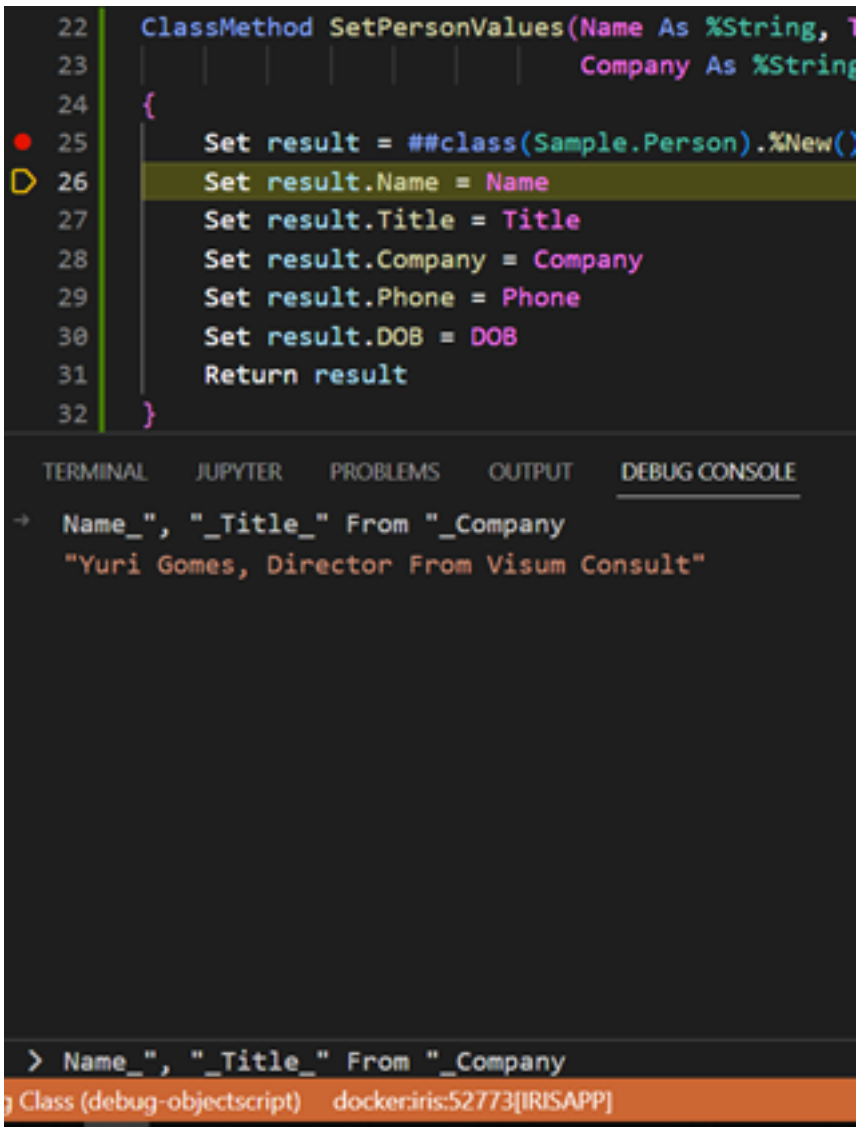


18. At last, the footer bar is orange, and it is indicating debugging:

Using Debug Console

Debug Console allows you to write expressions during Debug, so you can check values of different variables (simple type or object type) or validate the current value and other conditions. It can execute anything that can return some values, so $zv will work as well. Follow these steps to see some options:

1. Write Name", "Title" From "Company on Debug Terminal prompt and see Name, Tittle and Company concatenated values as output:



Using Debug Toolbar

Debug Toolbar allows you to control the execution of debugging, so check out button functions here:

1. Continue button (F9): continues the execution to the next breakpoint. If there is no breakpoint to go, it keeps on going to the end.
2. Step Over (F8): go to the next line without entering an internal method source code (do not enter Populate method).
3. Step Into (F7): go to the next line inside the internal method source code (go to the first line of the Populate method).
4. Step Out (Shift + F8): go to the current line of the caller source code of a method.
5. Restart (Crtl + Shift + F5): replay the debugging.
6. Stop: stops the debugging session.

Debug %CSP.REST class methods

To debug REST class methods, it is necessary to apply a little trick revealed by Fábio Gonçalves (check the article here https://community.intersystems.com/post/atelier-debugging-attach-process). It is necessary to add a HANG (my recommendation is between 20 and 30 seconds). Just follow these steps:

1. Open the File src\dc\Sample\PersonREST.cls and go to GetInfo() method, add HANG 30 (this is necessary to get time, 30 seconds, to start debugging of this method):

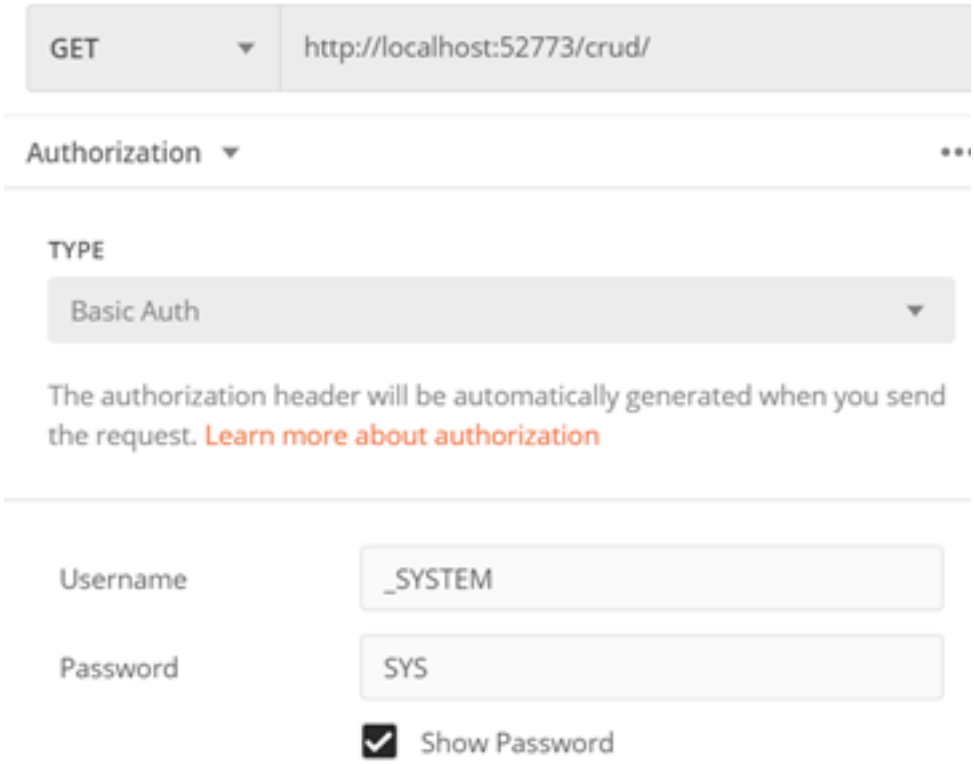
```
30  /// PersonsREST general information
    Debug this method
31  ClassMethod GetInfo() As %Status
32  {
33    HANG 30 //you need give time (30 seconds) to the Developer select the REST process to debug
34    SET version = ..#Version
35    SET info = {
36      "version": (version)
37    }
38    RETURN ..%ProcessResult($$$OK, info)
39  }
```

2. Set breakpoints on HANG and SET version lines (red circles):

```
30  /// PersonsREST general information
    Debug this method
31  ClassMethod GetInfo() As %Status
32  {
33    HANG 30 //you need give time (30 seco
34    SET version = ..#Version
35    SET info = {
36      "version": (version)
37    }
38    RETURN ..%ProcessResult($$$OK, info)
```

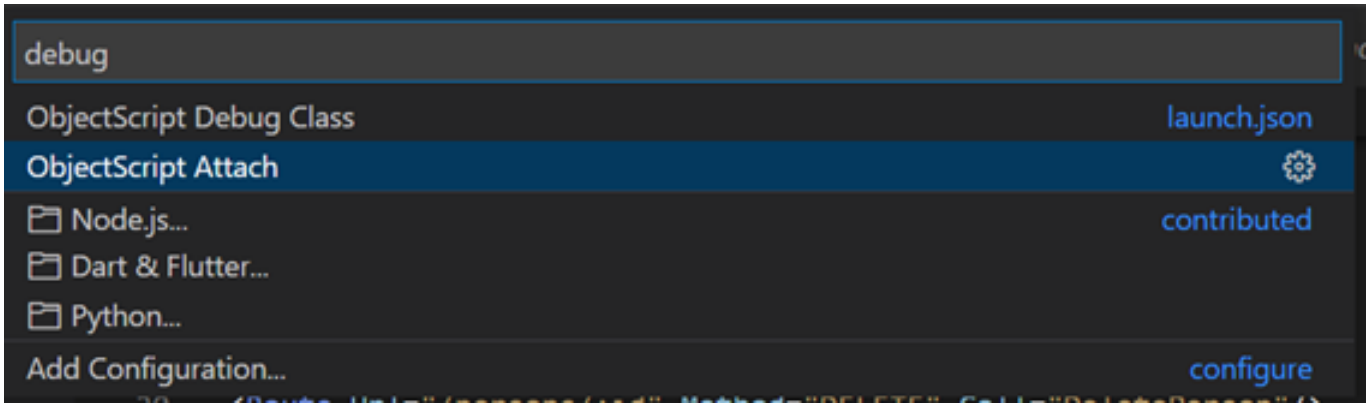
3. Open an HTTP client to call a REST method (I will call GetInfo from the class PersonREST):

4. Set Basic Auth to Authorization (Username SYSTEM and Password SYS):

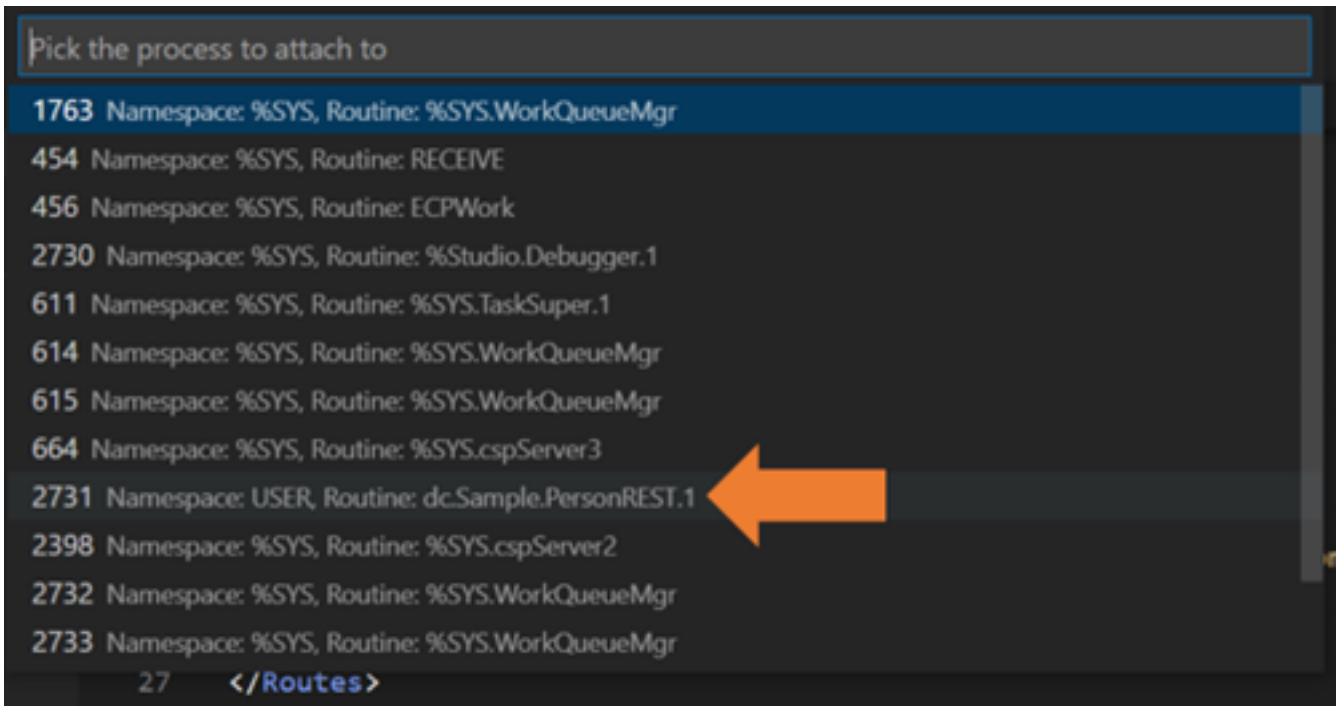


5. Send a GET http://localhost:52773/crud/.
6. Click on ObjectScript Attach on the footer bar:

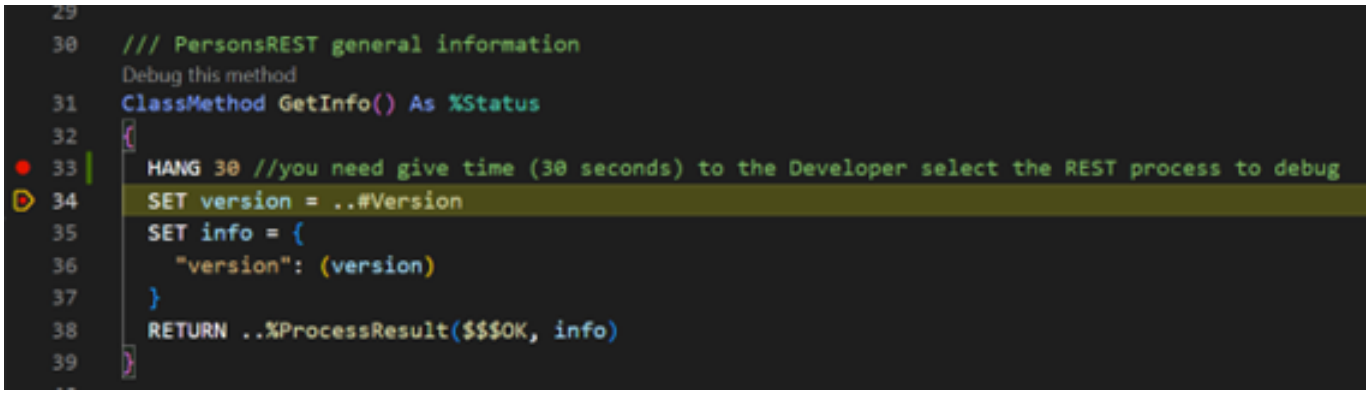
7. Choose ObjectScript Attach:



8. Choose the PersonREST process:



9. Wait for some time (about 30 seconds) for VSCode break the current execution on the breakpoint:



Note 1: remove HANG 30 when you finish your debugging, because HANG will pause the execution.

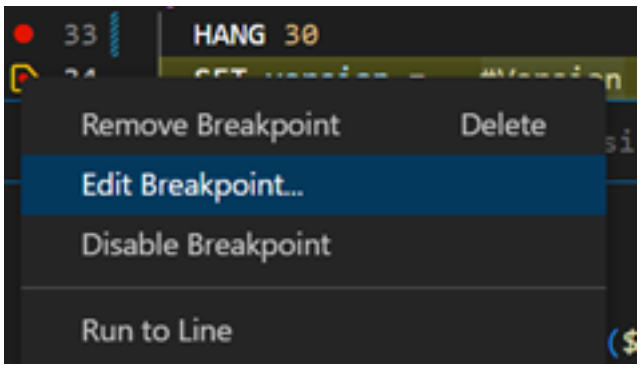
Note 2: if the PersonREST process does not appear, restart your VSCode, and try again.

10. Now you can do your debugging process.

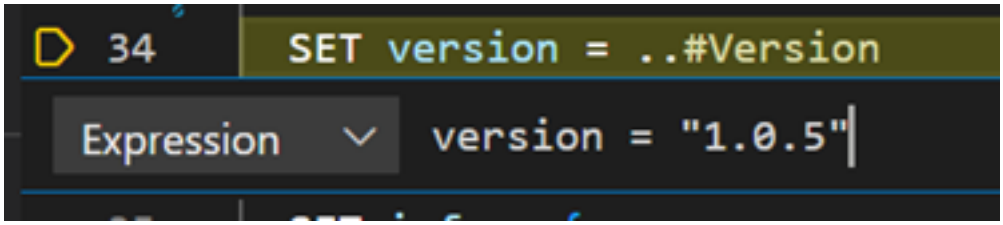
Edit breakpoint

You can click-right on the breakpoint to remove or configure conditional breakpoints (break on the breakpoint with a condition is true). The break command embedded right in the code will also be caught by the debugger. You can try this out as well.

1. Right-click on the breakpoint and select Edit Breakpoint:



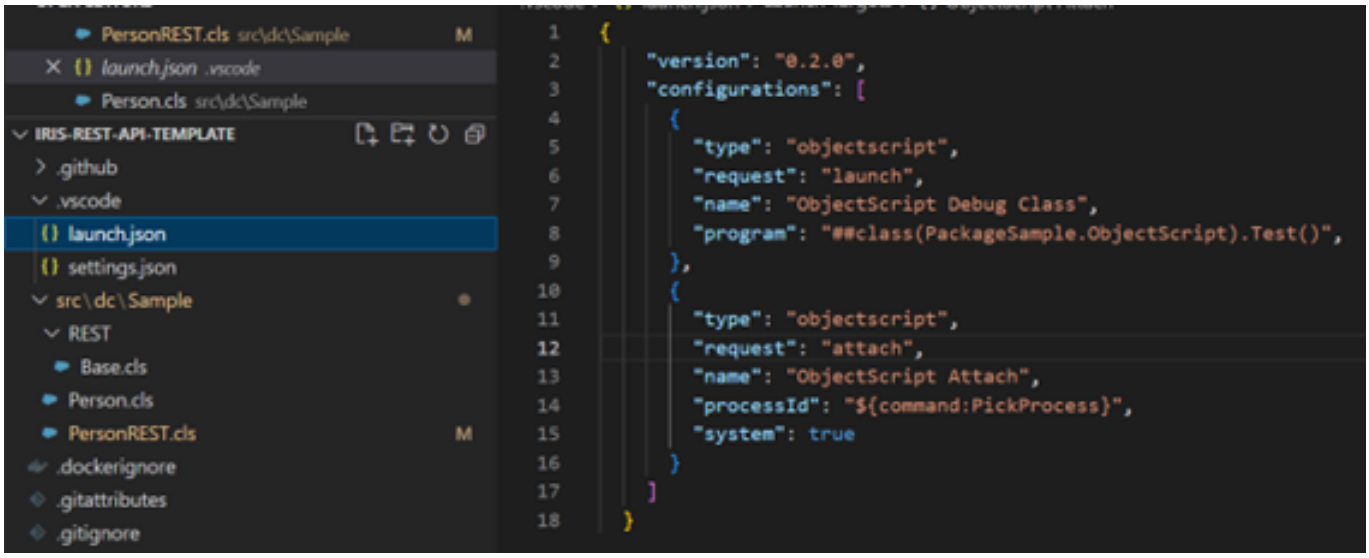
2. Enter the expression mentioned below and press Enter:



Note: try version = 1.0.5 to see breakpoint false and 1.0.6 to see breakpoint true.

Configure debugging option on .vscode/launch.json file

Launch.json file is used to configure debugging options for your project. To see the alternatives go to this file:



For this sample we have two options to select from to debug:

1. We can launch debugging on the class configured on the program attribute directly with a program (in this example the debugger will start in the PackageSample.ObjectScript class, method Test().
2. We can attach PickProcess to the selected program process you want to debug. In this example the debugger will open at the top a list of processes to select which process (each program has its process) will be debugged. The second option is more common.

In a general way, these attributes are mandatory for any debugging configuration (source: https://intersystems-community.github.io/vscode-objectscript/rundebug/):

- type - Identifies the type of debugger to use. In this case, objectscript is supplied by the InterSystems ObjectScript extension.
- request - Identifies the type of action for this launch configuration. Possible values are launch and attach.
- name - An arbitrary name to identify the configuration. This name appears in the Start Debugging drop-down list.

In addition, for an ObjectScript launch configuration, you need to supply the attribute program, which specifies the routine or ClassMethod to run when launching the debugger, as shown in the following example:

```
"launch": {
    "version": "0.2.0",
    "configurations": [
       {
        "type": "objectscript",
        "request": "launch",
        "name": "ObjectScript Debug HelloWorld",
        "program": "##class(Test.MyClass).HelloWorld()",
      },
      {
        "type": "objectscript",
        "request": "launch",
        "name": "ObjectScript Debug GoodbyeWorld",
        "program": "##class(Test.MyOtherClass).GoodbyeWorld()",
      },
   ]
}
```

For an ObjectScript attach configuration, you may supply the following attributes:

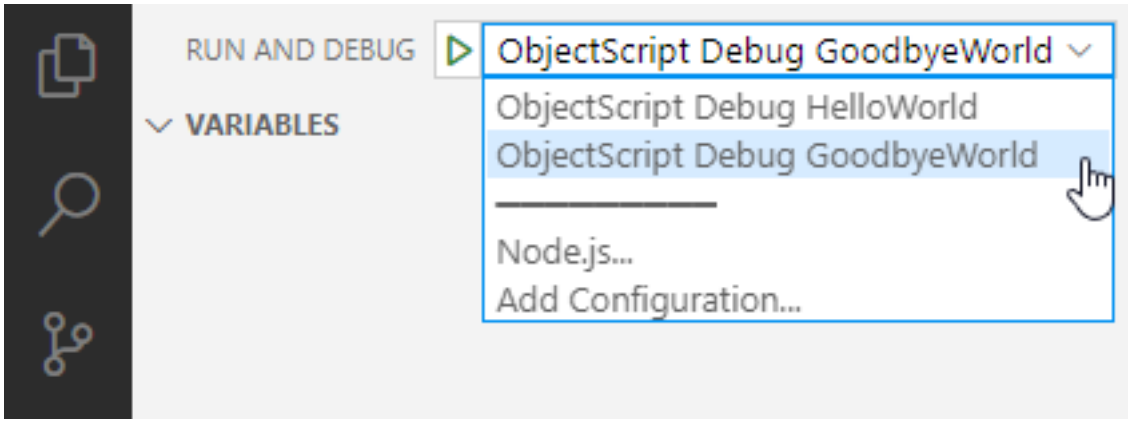
- processId - Specifies the ID of the the process to attach to a string or number. Defaults to "${command:PickProcess}" provide a drop-down list of process IDs to attach at runtime.
- system - Specifies whether or not to allow attachments to the system process. Set defaults to false.

The following example shows multiple valid ObjectScript attach configurations:

```
"launch": {
    "version": "0.2.0",
    "configurations": [
        {
            "type": "objectscript",
            "request": "attach",
            "name": "Attach 1",
            "processId": 5678
        },
        {
            "type": "objectscript",
            "request": "attach",
            "name": "Attach 2",
            "system": true
        },
    ]
}
```

Now, you can select a debugging configuration from the list VS Code provides in the Run and Debug field at the

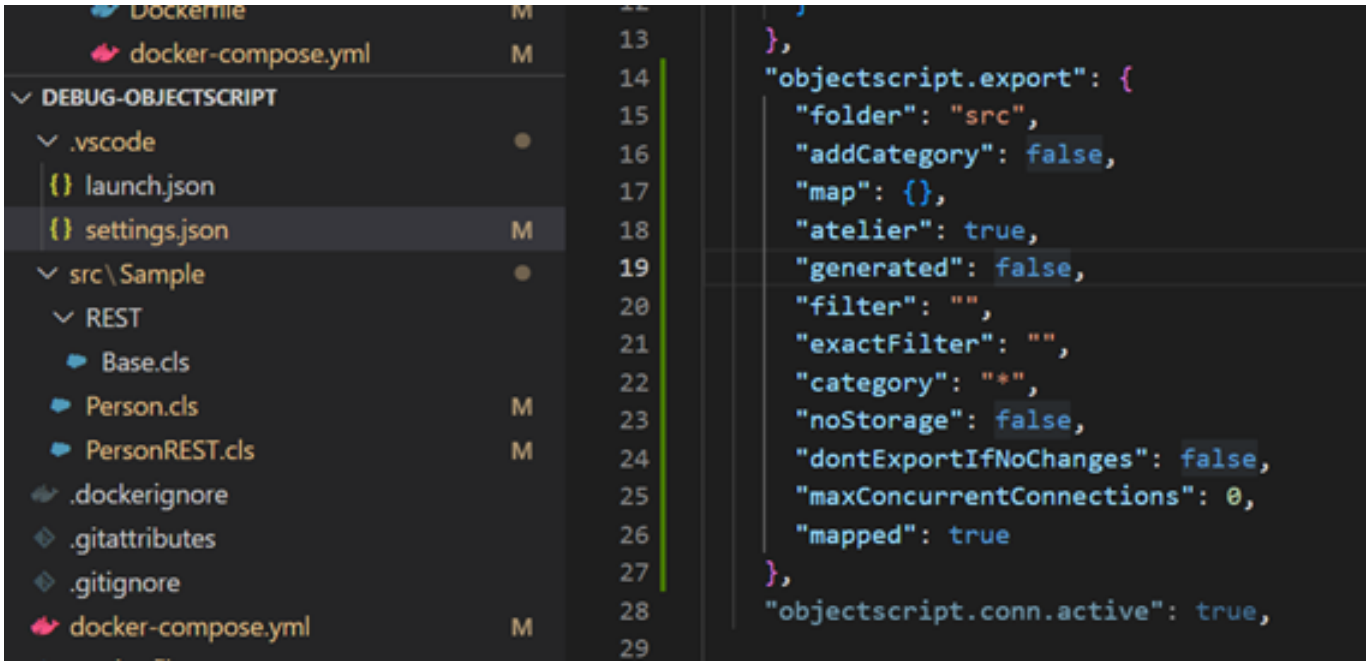
top of the debug sidebar (source: https://intersystems-community.github.io/vscode-objectscript/rundebug/):



If you click on the green arrow, the currently selected debugging configuration will run.
When starting ObjectScript launch debugging session, make sure that the file containing the program that you are debugging is open in your editor and is in the active tab. VS Code will start a debugging session with the server of the file in the active editor (the tab that the user is focused on). This also applies to attached ObjectScript debugging sessions.
This extension uses WebSockets to communicate with the InterSystems server during debugging. If you are experiencing issues when trying to start a debugging session, check ifthe InterSystems server ' s web server allows WebSocket connections.
Debugging commands and items on the Run menu function pretty much the same way they do for other languages supported by VS Code. For additional information on VS Code debugging, see the documentation resources listed at the start of this section.

Set the synchronization between a local source code and a server source code (into IRIS Server)

It is also important to have a folder structure for the source code, which can be adopted by "objectscript.export" settings. This information will be used to convert the server's class names to local files. If it's incorrect, it may open classes in the read-only mode even if they exist only locally.
1. Open .vscode\settings.json file and configure objectscript.export:



2. Hover the mouse over the folder, and add Category and other parameters to see documentation about it.

Other techniques to complement debugging process

In addition to debugging your program, remember to document your source code well, log operations performed by important code, and perform unit tests and source code static analysis tools. In this way, the quality of your programs will be much better, reducing maintenance and adjustment time and cost.

Learning more

You can read more about debugging ObjectScript on VSCode if you review these resources:

1. https://www.youtube.com/watch?v=diLHwA0rlGM
2. https://intersystems-community.github.io/vscode-objectscript/rundebug/
3. https://code.visualstudio.com/docs/introvideos/debugging
4. https://code.visualstudio.com/docs/editor/debugging
5. https://docs.intersystems.com/iris20221/csp/docbook/Doc.View.cls?KEY=TOSVSCode

#Best Practices #Debugging #InterSystems IRIS #VSCode
