Use Visual Studio Code as a Development Environment for InterSystems Applications

2024-05-06

InterSystems Corporation   1 Memorial Drive   Cambridge MA 02142   www.intersystems.com

Use Visual Studio Code as a Development Environment for InterSystems Applications
InterSystems IRIS Data Platform 2024-05-06
Copyright © 2024 InterSystems Corporation
All rights reserved.

InterSystems®, HealthShare Care Community®, HealthShare Unified Care Record®, IntegratedML®, InterSystems Caché®, InterSystems Ensemble® InterSystems HealthShare®, InterSystems IRIS®, and TrakCare are registered trademarks of InterSystems Corporation. HealthShare® CMS Solution Pack™ HealthShare® Health Connect Cloud™, InterSystems IRIS for Health™, InterSystems Supply Chain Orchestrator™, and InterSystems TotalView™ For Asset Management are trademarks of InterSystems Corporation. TrakCare is a registered trademark in Australia and the European Union.

All other brand or product names used herein are trademarks or registered trademarks of their respective companies or organizations.

This document contains trade secret and confidential information which is the property of InterSystems Corporation, One Memorial Drive, Cambridge, MA 02142, or its affiliates, and is furnished for the sole purpose of the operation and maintenance of the products of InterSystems Corporation. No part of this publication is to be used for any other purpose, and this publication is not to be reproduced, copied, disclosed, transmitted, stored in a retrieval system or translated into any human or computer language, in any form, by any means, in whole or in part, without the express prior written consent of InterSystems Corporation.

The copying, use and disposition of this document and the software programs described herein is prohibited except to the limited extent set forth in the standard software license agreement(s) of InterSystems Corporation covering such programs and related documentation. InterSystems Corporation makes no representations and warranties concerning such software programs other than those set forth in such standard software license agreement(s). In addition, the liability of InterSystems Corporation for any losses or damages relating to or arising out of the use of such software programs is limited in the manner set forth in such standard software license agreement(s).

THE FOREGOING IS A GENERAL SUMMARY OF THE RESTRICTIONS AND LIMITATIONS IMPOSED BY INTERSYSTEMS CORPORATION ON THE USE OF, AND LIABILITY ARISING FROM, ITS COMPUTER SOFTWARE. FOR COMPLETE INFORMATION REFERENCE SHOULD BE MADE TO THE STANDARD SOFTWARE LICENSE AGREEMENT(S) OF INTERSYSTEMS CORPORATION, COPIES OF WHICH WILL BE MADE AVAILABLE UPON REQUEST.

InterSystems Corporation disclaims responsibility for errors which may appear in this document, and it reserves the right, in its sole discretion and without notice, to make substitutions and modifications in the products and practices described in this document.

For Support questions about any InterSystems products, contact:

InterSystems Worldwide Response Center (WRC)
Tel: +1-617-621-0700
Tel: +44 (0) 844 854 2917
Email: support@InterSystems.com

Table of Contents

List of Tables

1

Introduction

Visual Studio Code (VS Code) is a free source code editor made by Microsoft for Windows, Linux and macOS. It provides built-in support for JavaScript, TypeScript and Node.js. You can add extensions to provide support for numerous other languages such as C++, C#, Java, Python, PHP, and Go, and runtimes such as .NET and Unity.

The InterSystems® extensions enable you to use VS Code to connect to an InterSystems IRIS® server and develop code in ObjectScript. This document covers issues specific to those extensions and working with ObjectScript and an InterSystems IRIS server. The Visual Studio Code Documentation is an excellent resource on VS Code, so it is a good idea to be familiar with it in addition to this document.

Development in ObjectScript involves both your local client machine, and an InterSystems IRIS server. Because both resources are required, the available workflows are different from that typical for many languages:

- With client-side editing, source code files are edited on the client, and saved to the local disk where they can be managed with a version control system. In addition, source files are imported into an InterSystems IRIS server, where they can be compiled, run, and debugged.
- With server-side editing, source code files can be edited directly on the InterSystems IRIS server. VS Code's multi-root workspace feature allows you to edit source files stored within different namespaces and even different servers simultaneously. Source files can then be compiled, run, and debugged on their respective servers.

Source code files are edited on the client, and saved to the local disk where they can be managed with a Version Control System. In addition, source files are exported to an InterSystems IRIS server, where they can be compiled, run, and debugged.

For existing customers, the InterSystems ObjectScript extension supports Studio extensions, as provided by `%Studio.Extension.Base`. If you rely on Studio extensions such as source control hooks, you can continue to use them in VS Code. VS Code is supported by InterSystems Caché® and Ensemble 2016.2 and higher, and all versions of InterSystems IRIS.

2

Installation

2.1 Install VS Code

Installation media for Visual Studio Code (VS Code) is available on the Visual Studio Code download page. Simply download the appropriate build for your platform and follow the installation procedure.

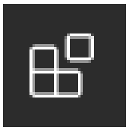
2.2 Install the InterSystems ObjectScript Extensions

To develop ObjectScript code using VS Code, you must install the following extensions:

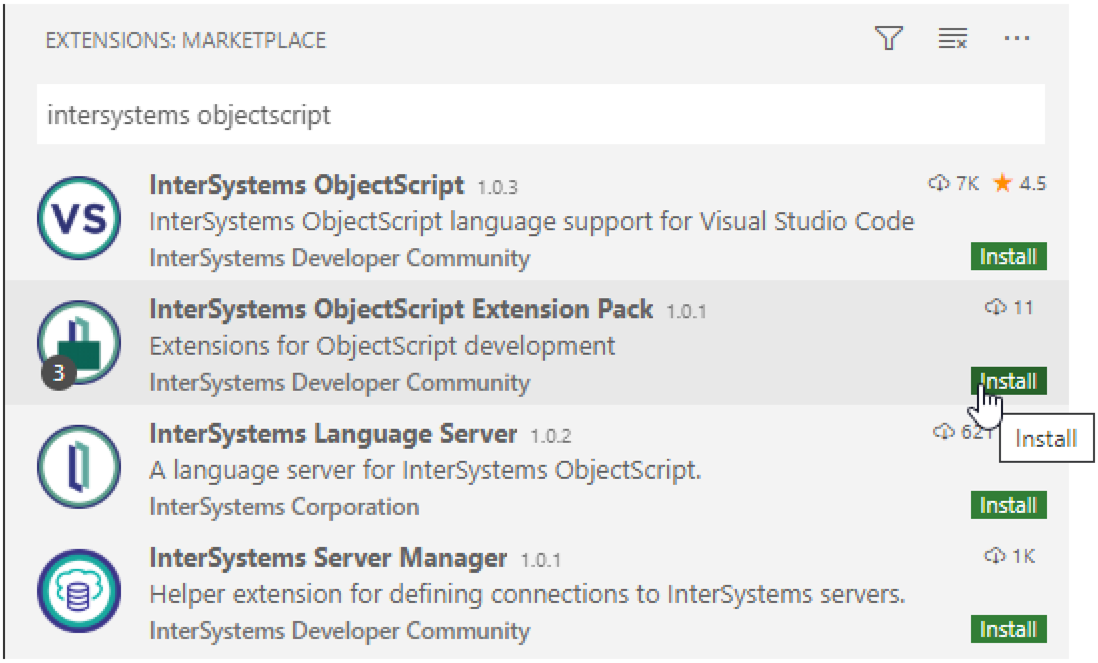
- InterSystems Language Server
- InterSystems ObjectScript
- InterSystems Server Manager

These extension are grouped together in the InterSystems ObjectScript Extension Pack. To install these extensions:

1. Run VS Code
2. From within the application, click the extensions button in the Activity Bar on the left edge of the VS Code window:

3. Type `intersystems` in the search field to find the relevant extensions in the Marketplace, as illustrated in the following image:



intersystems objectscript

InterSystems ObjectScript 1.0.3 — 7K ★ 4.5
InterSystems ObjectScript language support for Visual Studio Code
InterSystems Developer Community
Install

InterSystems ObjectScript Extension Pack 1.0.1 — 11
Extensions for ObjectScript development
InterSystems Developer Community
Install

InterSystems Language Server 1.0.2 — 621
A language server for InterSystems ObjectScript.
InterSystems Corporation
Install

InterSystems Server Manager 1.0.1 — 1K
Helper extension for defining connections to InterSystems servers.
InterSystems Developer Community
Install

4. Select the **Install** button for the **InterSystems ObjectScript Extension Pack**. Alternatively, you can install each extension independently by selecting the **Install** button for each of the required extensions.

Note: If you are using an Nginx web server to connect to InterSystems IRIS, you must configure Nginx to use VS Code with the InterSystems ObjectScript Extension Pack.

Note: If you are using a Microsoft IIS web server to connect to InterSystems IRIS, you must ensure the WebDAV module is disabled to use the InterSystems ObjectScript Extension Pack. To debug your code with VS Code, you must enable the WebSockets feature. See Configure IIS to Use VS Code.

3

Migrate from Studio

The extensions that make up the InterSystems ObjectScript Extension Pack contain many useful features that make migrating from InterSystems Studio easy. This page describes some of these features.

3.1 Server-Side Editing

You can configure VS Code to edit code directly on a server, which is analogous to Studio's architecture. However, VS Code enhances this workflow with support for having multiple server-namespaces open at the same time (using VS Code's multi-root workspace feature) and for filtering the files shown for each server-namespace. See the Server-side Editing for more information on how to configure this feature.

3.2 Server-Side Source Control

VS Code supports server-side source control without requiring any additional configuration. Server-side source control is supported for both server-side and client-side editing. If a source control class is active, its hooks fire automatically for document lifecycle events like creation, first edit, save and deletion. You can access the server source contol menu in these locations:

- The source control icon (when a document is open)
- An open document's context menu
- A node in the context menu for the ObjectScript Explorer
- A node in the context menu for the VS Code Explorer.

3.3 Server-Side Projects

VS Code supports using existing Studio projects, as well as the creation, modification and deletion of them. See Work with Projects for more information about this feature and how to use it.

3.4 Accurate Syntax Coloring

The InterSystems Language Server extension leverages VS Code’s semantic tokens API to provide the same accurate syntax coloring for InterSystems ObjectScript and other embedded languages that Studio users are familiar with. For more information on how to customize the syntax colors for InterSystems tokens, see the Language Server’s README. InterSystems provides a command for Windows users to migrate their existing color customizations from Studio.

3.5 Import Server Definitions Command

The InterSystems Server Manager extension provides the **Import Servers from Windows Registry** command, which will import any Studio server defintions from your Windows registry into VS Code so you can continue using them without having to do the migration youself.

To invoke the command, open the command palette, find the **InterSystems Server Manager: Import Servers from Windows Registry** menu option, and run it.

3.6 Load Studio Snippets Command

The InterSystems ObjectScript extension provides the **Load Studio Snippets** command, which will load any user defined snippets from Studio into VS Code. It works by reading the locations of Studio user defined snippets files from the Windows registry, converting the snippets contained in those files to VS Code’s JSON format and lastly writing the snippets to a new global snippets file called isc-studio.code-snippets. This command will only convert snippets for ObjectScript, Class Definition Language (UDL) or HTML; all other snippets are ignored.

To invoke the command, open the command palette, find the **ObjectScript: Load Studio Snippets** menu option, and run it.

After you load your snippets, InterSystems recommends opening the generated file and enhancing the snippets so that they can fully leverage features available in VS Code which Studio does not support, like tabstops and variable substitution.

You can open the snippets file in two ways:

- Upon loading your snippets, select the **Open File** button in the success notification box.
- Select the Settings button. ⚙ Select the **Configure User Snippets** from the settings menu, and then select the `isc-studio.code-snippets` file from the drop-down menu.

3.7 Load Studio Syntax Colors Command

The InterSystems ObjectScript extension provides the **Load Studio Syntax Colors** command, which will load the editor background and syntax foreground colors from Studio into VS Code. It works by reading the color configurations from the Windows registry and storing them in VS Code’s User Settings as customizations of one of the InterSystems default themes provided by the Language Server extension. The command uses the background color loaded from Studio to determine which default theme it should modify, and will activate the modified theme automatically. This command will not load foreground colors for any syntax tokens that have a custom background color because per-token background colors are not supported in VS Code. This command requires that the InterSystems Language Server extension is installed and active.

To invoke the command, open the command palette, find the **ObjectScript: Load Studio Syntax Colors** option, and run it.

3.8 New File Commands

The InterSystems ObjectScript extension provides commands for creating new Interoperability classes. Commands are provided for Business Operation, Business Process, Business Rule, Business Service and Data Transformation classes. These commands are modeled after the wizards in Studio’s **File** > **New...** > **Production** menu.

In VS Code, these commands are available by navigating to **File** > **New File...**, or by selecting the link on the **Get Started** welcome page.

3.9 Keyboard Shortcuts

In general, VS Code keyboard shortcuts are entirely customizable, as described in the VS Code documentation. However, VS Code is pre-configured with a number of shortcuts that match Studio. Download a cheat sheet here.

This section provides mapping tables for Studio users to compare Studio shortcuts with the shortcuts which VS Code provides by default.

Table 3–1: General

Studio	VS Code	Action	VS Code Notes
F8	**F11**	Toggles full screen display of IDE window	
Ctrl+N	**Ctrl+N**	New document	
Ctrl+O	**Ctrl+O**, **Ctrl+P**	Open document	To search for files which begin with the percent character (%) using Quick Open (**Ctrl+P**), you must precede the file name with a space. To search for files by name using VS Code's Quick Open menu when you are editing code on the server-side, you must enable the Proposed APIs.
Ctrl+Shift+O	**Ctrl+K, Ctrl+O**	Open project	Opens a folder on-disk. If you are not using client-side source control, opens a project from the ObjectScript pane.
Ctrl+P	**Ctrl+P**	Print	
Ctrl+S	**Ctrl+S**	Save	
Ctrl+Shift+I		Export	For client-side editing, use the **Export Code from Server** command from the command palette or export from the ObjectScript Explorer
Ctrl+I		Import local	For client-side editing, files are imported when saved by default. You can also use the **Import and Compile** command in the Explorer context menu. For server-side editing, right-click on an `isfs` workspace folder and select the **Import Local Files...** command

Table 3–2: Display

Studio	VS Code	Action	VS Code Notes
Ctrl++	**Ctrl+K** then **Ctrl+J**, **Ctrl+K** then **Ctrl+0**	Expand, collapse all	
Ctrl+Left Arrow select plus icon	**Ctrl+K** then **Ctrl+], Ctrl+K** then **Ctrl+[**	Expand, collapse all block sections	
Ctrl+Shift+V	**Ctrl+Shift+V**	View other documents related to the current document (such as MAC or INT routines)	
Alt+2	**Ctrl+Shift+U**	Toggle output window display	
Alt+5		Toggle display of **Code Snippets** window	VS Code includes no UI for code snippets
Alt+6	**Ctrl+Shift+F**	Toggle display of **Find in Files** window	
Ctrl+Alt+, +Ctrl+Alt+ -	**Ctrl+ +**, **Ctrl+ -**	Increase, decrease font	
Ctrl+Alt+Space	**Ctrl+Shift+P** and begin typing `render`	Toggle display of whitespace symbols, spaces, new lines, and tabs	
Ctrl+B	Always on	Toggle bracket matching	
Ctrl+Tab	**Ctrl+Shift+]**	Next window	
Ctrl+Shift+Tab	**Ctrl+Shift+[**	Previous window	

Table 3–3: Navigation

Studio	VS Code	Action	VS Code Notes
Home, **End**	**Home**, **End**	Go to beginning, end of line	
Ctrl+Home, **Ctrl+End**	**Ctrl+Home**, **Ctrl+End**	Go to beginning, end of document	
Ctrl+ -, **Ctrl+Shift+ -**	**Alt+Left Arrow**, **Alt+Right Arrow**	Go back, forward	
PgUp, PgDn	**PgUp, PgDn**	Page Up, Down	
Ctrl+PgUp, **Ctrl+PgDn**	**Alt+PgUp**, **Alt+PgDn**	Go to top, bottom of page	
Ctrl+Down Arrow, **Ctrl+Up Arrow**	**Ctrl+Down Arrow**, **Ctrl+Up Arrow**	Scroll down, up	
Ctrl+G	**Ctrl+Shift+O**	Go to	You can use **Ctrl+T** to find a symbol across files. More information can be found in the VS Code documentation
Ctrl+F3, **Ctrl+Shift+F3**	**F8**, **Shift+F8**	Go to next, previous error	
Alt+F3, **Alt+Shift+F3**	**F8**, **Shift+F8**	Go to next, previous warning	
Ctrl+]	**Ctrl+Shift+\**	Go to bracket	

Table 3–4: Editing

Studio	VS Code	Action	VS Code Notes
Ctrl+Delete		Delete next word or delete to end of current word	To reproduce this capability, try an extension such as Emacs Friendly Keymap
Ctrl+Backspace or **Ctrl+Shift+Delete**		Delete previous word or delete to start of current word	To reproduce this capability, try an extension such as Emacs Friendly Keymap
Ctrl+Shift+L	**Ctrl+Shift+K**	Delete line	
Ctrl+C or **Ctrl+Insert**	**Ctrl+C**	Copy	
Ctrl+X or **Shift+Delete**	**Ctrl+X**	Cut	
Ctrl+L	**Ctrl+X**	Cut line	
Ctrl+V or **Shift+Insert**	**Ctrl+V**	Paste	
Ctrl+A	**Ctrl+A**	Select all	
Ctrl+Z, **Ctrl+Y** or **Ctrl+Shift+Z**	**Ctrl+Z**, **Ctrl+Shift+Z**	Undo, redo	

Studio	VS Code	Action	VS Code Notes
Ctrl+Space	**Ctrl+Space**	Show Studio Assist popup or trigger code completion	In VS Code, code completion suggestions appear automatically as you type
Ctrl+~		Toggle tab expansion	Select **Spaces** in status bar
Ctrl+U, **Ctrl+Shift+U**		Uppercase, lowercase selection	To reproduce this capability, try an extension such as change-case
Ctrl+Alt+U		Titlecase (initial caps) selection	To reproduce this capability, try an extension such as change-case
Ctrl+(	(	Insert open and close parentheses. (Does not work on German and Swiss keyboard layouts)	To reproduce this capability, try an extension such as change-case
Ctrl+{	{	Insert open and close braces	
Ctrl+=	**Ctrl+Shift+P** and begin typing `format`	Indentation cleanup — cleans up indentation on a selected block of text	
Ctrl+/, **Ctrl+Shift+/**	**Ctrl+/**	Comment, uncomment line or block of text	
Ctrl+Alt+/, **Ctrl+Shift+Alt+/**	**Ctrl+/**	Add, remove comment markers from block of text	
Ctrl+E		In an ObjectScript document, commands in a selection are replaced with their full names	With the InterSystems Language Server installed, you can configure its formatter to expand command names and then format some or all of your document
Ctrl+Shift+E		Compress commands	With the InterSystems Language Server installed, you can configure its formatter to contract command names and then format some or all of your document

Table 3–5: Find and Replace

Studio	VS Code	Action	VS Code Notes
Ctrl+F, **Ctrl+H**	**Ctrl+F**, **Ctrl+H**	Find, replace	
F3, **Shift+F3**	**F3**, **Shift+F3**	Find next, previous	
Ctrl+Shift+F	**Ctrl+Shift+F**, **Ctrl+Shift+H**	Find, replace in files	
Ctrl+, (comma)	**Ctrl+P**	Search for class	
Ctrl+Shift+G, **Ctrl+Alt+G**	**Alt+Right Arrow**, **Alt+Left Arrow**	Go to, go back	

Table 3–6: Bookmarks

Studio	VS Code	Action	VS Code Notes
Ctrl+F2		Toggle bookmark on current line	To reproduce this capability, try a third-party extension
F2, **Shift+F2**		Go to next, previous bookmark	To reproduce this capability, try a third-party extension
Ctrl+Shift+F2		Clear all bookmarks	To reproduce this capability, try a third-party extension

Table 3–7: Build and Compile

Studio	VS Code	Action	VS Code Notes
F7	**Ctrl+Shift+F7**	Rebuilds all documents in project	
Ctrl+F7	**Ctrl+F7**	Compile active document	
Ctrl+Shift+F7		Compile with options	Execute the **Import and Compile Current File with Specified Flags...** from the command palette

Table 3–8: Debugging

Studio	VS Code	Action	VS Code Notes
Ctrl+Shift+A		Debug: attach	See Running and Debugging for how to debug a running process
F9	**F9**	Debug: toggle breakpoint on current line	
Ctrl+F5, **Shift+F5**	**F5**, **Shift+F5**	Debug: start, stop	
Ctrl+Shift+F5	**Ctrl+Shift+F5**	Debug: restart	
F11, **Shift+F11**	**F11**, **Shift+F11**	Debug: step into, out	
F10	**F10**	Debug: step over	

4

Get Acquainted with the User Interface for the InterSystems VS Code Extensions

When you install the InterSystems ObjectScript Extension Pack for Visual Studio Code (VS Code), it adds additional capability to the VS Code user interface to support development in ObjectScript. This page summarizes those added capabilities.

For more information about the features of VS Code's standard UI, refer to the section User Interface in the VS Code documentation.

4.1 Explorer View Context Menu Enhancements

The Explorer view is a standard VS Code view which allows you to view and organize directories and files within your workspaces. Access the Explorer view by selecting the Explorer button ⧉ from the Activity Bar.

The InterSystems extension pack adds the following items to context menus in this view:

- **Add Server Namespace to Workspace...** in the context menu of folders
- **Import and Compile** in the context menu of folders and files when you are connected to an InterSystems server
- **Import Without Compilation** in context menu of folders and files when you are connected to an InterSystems server

4.2 InterSystems View Container

The InterSystems extension pack provides most of the capabilities necessary to support development in ObjectScript within three views:

- the InterSystems Servers view, which you can use to manage connections to your InterSystems servers and add server-side directories to your workspace
- the ObjectScript Explorer view, which you can use to explore files on a connected InterSystems server.
- the InterSystems Projects Explorer view, which you can use to create, manage, and explore projects. For more information on using this view, see Work with Projects.

All three of these views are available within the InterSystems view container. To access the InterSystems view container, select the InterSystems button in the Activity bar.

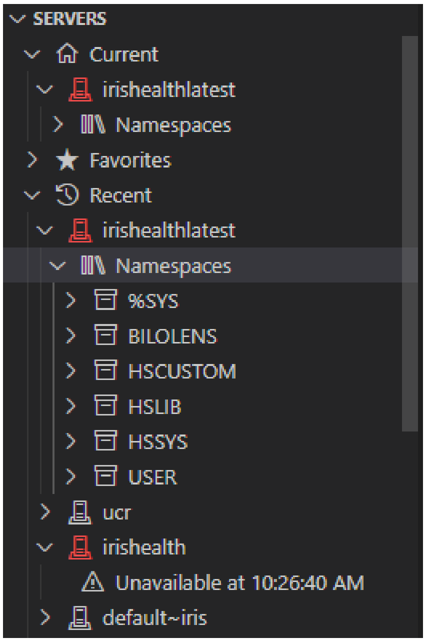
When there is no workspace or folder open in the Explorer, only the InterSystems Servers view is available. Connect to an InterSystems server or open a resource within the Explorer to display the ObjectScript Explorer and InterSystems Projects Explorer.

Note: Views within the InterSystems view container can be dragged to a different view container. This means, for example, that you could move the InterSystems Servers view into VS Code's main Explorer view container.

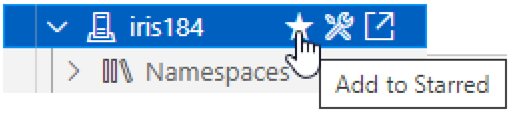
When a VS Code container has only a single view in it, the view header merges with the container header, with the two names separated by a colon.

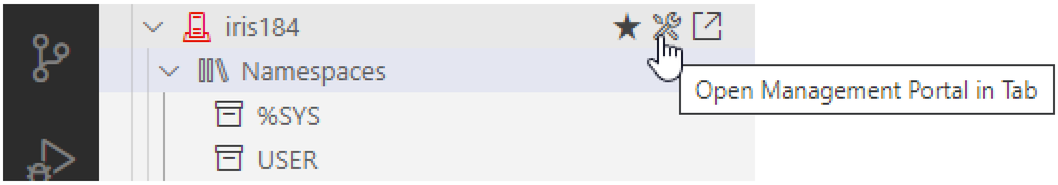
4.2.1 InterSystems Servers

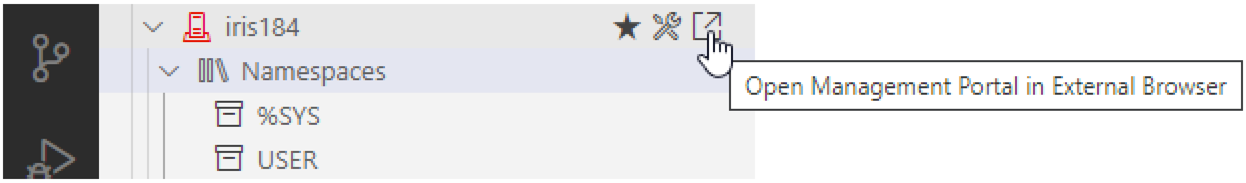
This view shows server resources in a tree format, as illustrated in the following screenshot:



This view groups servers into a variety of folders: servers currently in use, user-defined favorites, and servers which have been recently used. Within the view, you can perform operations on the servers. When you move the cursor over a server listing, command buttons appear which let you mark the server as a favorite, open the server's Management Portal in a VS Code tab in the built-in Simple Browser, or open its Management Portal in an external browser, as illustrated below

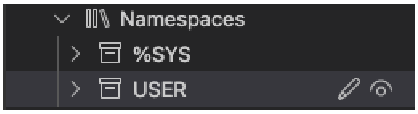






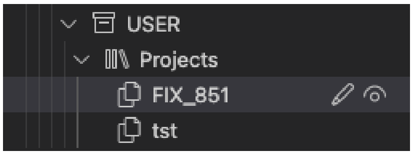
4.2.1.1 Viewing and Editing Source Code on the Server

To view and edit source code on a server, expand the target server in the tree view, then expand its **Namespaces** folder. Hover over the target namespace to reveal its command buttons, as illustrated in the following image:



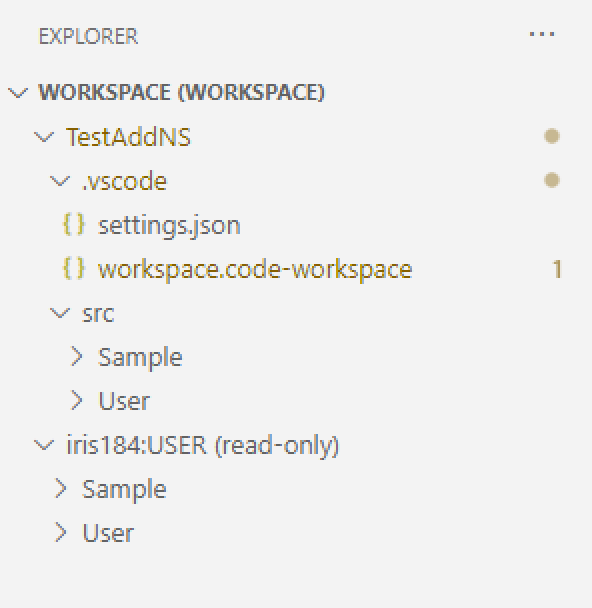
- Select the edit (pencil) button to add an `isfs://server:namespace/` folder to your VS Code workspace which corresponds with that namespace.
- Select the view (eye) button to add an `isfs-readonly://server:namespace/` folder to your VS Code workspace which corresponds with that namespace.
- Hold the **Alt** or **Option** key while selecting the edit or view button to add a folder that also gives you access to server-side web application files (such as .csp files).

If you want to add a folder that shows only a single project's contents, expand the target namespace and the *Projects* folder to reveal the projects in the target namespace. Hover over the target project to reveal its command buttons, as illustrated in the following image:



- Select the edit (pencil) button to add an `isfs://server:namespace/?project=projectName` folder to your VS Code workspace.
- Select the view (eye) button to add an `isfs-readonly://server:namespace/?project=projectName` folder to your VS Code workspace.

Once you have added a server-side namespace to the workspace, VS Code opens the Explorer view showing the added namespace. In the following image, the workspace includes the Sample and User packages in the src folder on the client, and the Sample and User packages in the USER namespace on the server, with read-only access:



EXPLORER ···

- WORKSPACE (WORKSPACE)
  - TestAddNS ●
    - .vscode ●
      - { } settings.json
      - { } workspace.code-workspace 1
    - src
      - Sample
      - User
  - iris184:USER (read-only)
    - Sample
    - User

To learn more about using isfs and isfs-readonly folders, see Server-Side Editing.

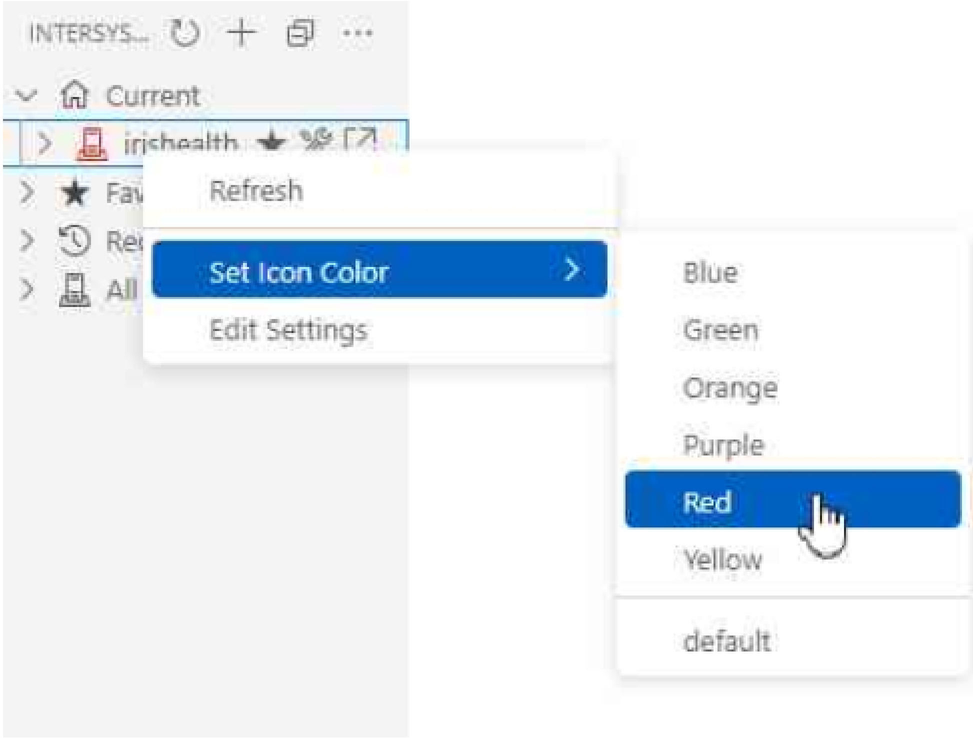
Important: If you are already doing client-side editing of your code (for example, to manage it with Git) be sure you understand the consequences of also doing server-side editing using isfs. If in doubt, limit yourself to isfs-readonly by only using the eye icon.

4.2.1.2 Adding a Server

You can use the plus sign (+) in the title bar of the server resource explorer pane to add a server as described in the section Configure a Server.

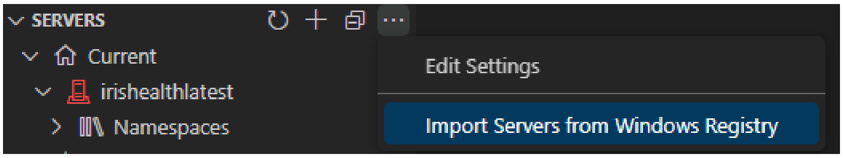
4.2.1.3 Server Context Menu

Servers listed in the InterSystems Server view provide a context menu which includes options to refresh the server, edit the settings.json specification for the server, and set the color of the icon for the server in the tree view, as illustrated in the following image:



4.2.1.4 Import Server Connections

On Windows, the Server Manager can create connection entries for all connections you previously defined with the original Windows app called InterSystems Server Manager. This action is available by selecting **Import Servers from Windows Registry** from the **...** (ellipsis) menu in the title bar of the server resource explorer pane, as illustrated in the following image:



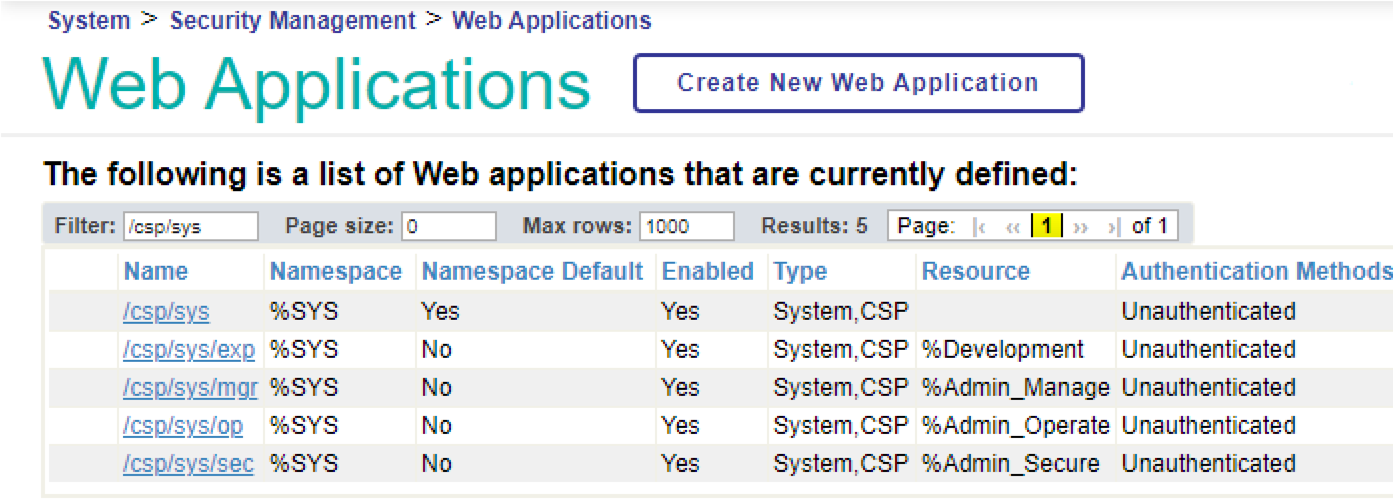
4.2.1.5 Notes About the VS Code Simple Browser

Only one Simple Browser tab can be open at a time, so launching a second server's Management Portal replaces the previous one.

If the server version is InterSystems IRIS 2020.1.1 or later you need to change a setting on the suite of web applications that implement Management Portal. The Simple Browser is not permitted to store the Portal's session management cookies, so the Management Portal must be willing to fall back to using the CSPCHD query parameter mechanism.

To do so:

1. In Management Portal, select *System Administration* > *Security* > *Applications* > *Web Applications*.
2. Enter `/csp/sys` in the *filter* field to find the web applications which have paths beginning with `/csp/sys`.



System > Security Management > Web Applications

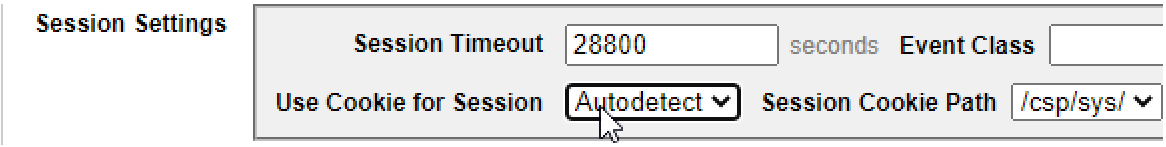
Web Applications

The following is a list of Web applications that are currently defined:

Filter: /csp/sys Page size: 0 Max rows: 1000 Results: 5 Page: 1 of 1

Name	Namespace	Namespace Default	Enabled	Type	Resource	Authentication Methods
/csp/sys	%SYS	Yes	Yes	System,CSP		Unauthenticated
/csp/sys/exp	%SYS	No	Yes	System,CSP	%Development	Unauthenticated
/csp/sys/mgr	%SYS	No	Yes	System,CSP	%Admin_Manage	Unauthenticated
/csp/sys/op	%SYS	No	Yes	System,CSP	%Admin_Operate	Unauthenticated
/csp/sys/sec	%SYS	No	Yes	System,CSP	%Admin_Secure	Unauthenticated

3. For each application:
   a. Select the link in the **Name** column to edit the application definition.
   b. In the section labeled **Session Settings**, change the value of **Use Cooke for Session** from **Always** to **Autodetect**.



Session Settings

Session Timeout: 28800 seconds

Event Class:

Use Cookie for Session: Autodetect

Session Cookie Path: /csp/sys/

   c. Select **Save** to save the change.

This change should not affect the use of session cookies on ordinary browsers.

4.2.2 ObjectScript Explorer

When a VS Code workspace is not connected to an InterSystems IRIS server, the ObjectScript Explorer provides a button that lets you select a server and namespace. Once the workspace is connected to an InterSystems server, the ObjectScript Explorer shows files on the server, grouped by type of file.

If the workspace is configured for server-side editing, the ObjectScript Explorer is not available. In this configuration, the Explorer view lists files on the server, not on the local machine, making the ObjectScript Explorer view irrelevant.

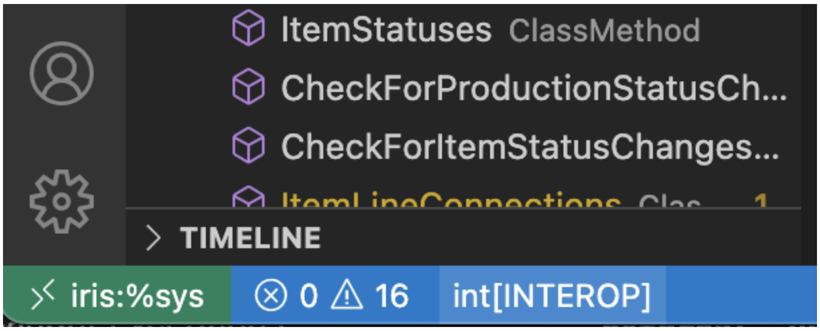
The ObjectScript Explorer provides the following items:

- **Compile** — Compiles files on the server
- **Delete** — Deletes files from the server
- **Export** — Exports files to the workspace on the client
- **Server Command Menu...** — Allows you to select a command from menus which are configured on the server.
- **Server Source Control...** — Allows you to select a command from menus which are configured on the server.

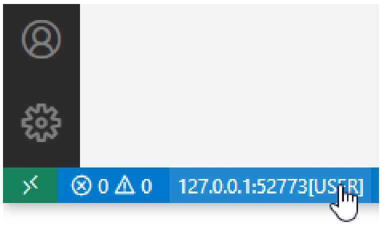
The InterSystems IRIS documentation section Extending Studio describes how to configure menus for source code control and other purposes. Entries from menus named %SourceMenu and %SourceContext appear in the **Server Source Control** menu provided the source control class does not disable the entry. For example, the source control class may disable checkout if the file is already checked out.

4.3 Server Connection Status Bar

The connection status of the current server can be seen in the VS Code status bar. If the server connection is active, the connected server and namespace will be shown in the status bar. If the server connection info is defined in the `intersystems.servers` settings object, the name of the server and namespace appears:

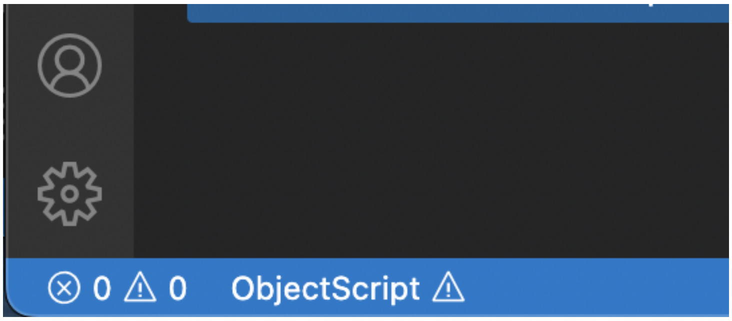


Otherwise, the host, port, and namespace appear:



Hover over the status bar item to see more detailed connection information, like a full connection URL and the username of the connection. Click on the status bar item to open the Server Actions menu. You can add custom entries to this menu.

If the server connection is inactive, the status bar displays the connection info or the word `ObjectScript` will be shown, along with an error or warning icon:



Hover over the status bar item to see more detailed error information. Click on the status bar item to open a menu that will help you activate your connection.

5

Configure the InterSystems VS Code Extensions

Visual Studio Code (VS Code) settings enable you to customize various aspects of its behavior. The InterSystems extensions provide settings used to configure VS Code for ObjectScript development.

5.1 Settings

Many available settings are general to VS Code, and you can learn about them in the Visual Studio Code Documentation. The InterSystems Server Manager, InterSystems ObjectScript, and InterSystems Language Server extensions supply additional settings which you can use to define InterSystems IRIS servers and connections to those servers.

VS Code stores these settings at several levels:

- User — User settings are stored in a file location specific to you and apply globally to any instance of VS Code or any VS Code workspace that you open.
- Workspace — a workspace is a set of directories you want to use when you’re working on a particular project. Workspace settings are stored in a file inside the workspace directory structure and apply to anyone who opens the workspace. See Configure Your Workspace for an InterSystems Project.
- Folder — If more than one folder is present in a workspace, you can select the folder where the settings file is stored by selecting from the Folder drop down list.

See the User and Workspace Settings section of the VS Code documentation for details.

For a list of all the settings which the InterSystems extensions contribute, see the Settings Reference.

5.2 Configure Your Workspace for an InterSystems Project

VS Code has a concept of a workspace, which is a set of directories you want to use when you’re working on a particular project. In the simplest setup when you are working within a single directory, a VS Code workspace is just the root folder of your project. In this case you keep workspace-specific settings in two files inside a .vscode directory located at the root of your project. Those two files are settings.json, which contains most configuration settings, and launch.json, which contains debugging configurations.

5.2.1 settings.json

Here is the simplest settings.json file content for an ObjectScript project (connection details are managed by the InterSystems Server Manager, as described in Configuring a Server):

```
{
    "objectscript.conn": {
        "server": "iris",
        "ns": "USER",
        "active": true
    }
}
```

If you need ObjectScript compilation flags other than the default ones, add an `objectscript.compileFlags` property to settings.json (more `compileFlags` information is available on the settings reference page):

```
{
    "objectscript.conn": {
        "server": "iris",
        "ns": "USER",
        "active": true,
    },
    "objectscript.compileFlags": "cuk/compileembedded=1"
}
```

5.2.2 launch.json

Here is the simplest launch.json file content, with which you can debug the method Test in the class Example.Service, passing 2 parameters as input (see Running and Debugging for more information):

```
{
    "version": "0.2.0",
    "configurations": [
        {
            "type": "objectscript",
            "request": "launch",
            "name": "Example.Service.Test",
            "program": "##class(Example.Service).Test(\"answer\",42)"
        }
    ]
}
```

If you want to debug a running process, launch.json should have a section like this, which will present a dropdown menu of running processes:

```
{
    "version": "0.2.0",
    "configurations": [
        {
            "type": "objectscript",
            "request": "attach",
            "name": "Example-attach-to-process",
            "processId": "${command:PickProcess}"
        }
    ]
}
```

Note that `configurations` is an array, so you can define multiple configurations and choose the one to use from a dropdown menu in the **Debug** pane.

5.2.3 Multi-Root Workspaces and Server-Side Editing

If your project requires more than a single root folder, you need to use a feature called multi-root workspaces. See Multi-Root Workspaces in the VS Code documentation.

In this case settings are stored in a file with a .code-workspace file extension. This workspace file can be located anywhere. It defines what root folders the workspace consists of, and may also store other settings that would otherwise be stored in settings.json or launch.json. Settings in a root folder's .vscode/settings.json or .vscode/launch.json will override those in the workspace file, so be careful to use one or the other unless you truly need to leverage this flexibility.

You can have a workspace file even if you are only working with a single root folder.

Important: If you are editing server-side code you *must* use a workspace file.

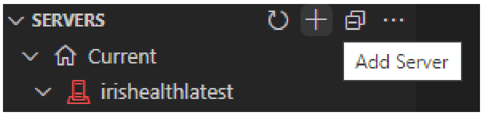
To edit InterSystems ObjectScript extension settings in a .code-workspace file in VS Code, perform the following steps:

1. Open the workspace using **File** > **Open Workspace from File**.
2. Select **File** > **Preferences** > **Settings** (**Code** > **Preferences** > **Settings** on Mac).
3. Select the **Workspace** tab.
4. Search for objectscript: conn and click on **Edit in settings.json**. This opens the .code-workspace file for that workspace.

The InterSystems Objectscript extension uses the multi-root workspaces feature to support Objectscript development directly in namespaces on InterSystems servers.

5.3 Configuring a Server

To add an InterSystems IRIS server, select the plus sign (+) in the title bar of the InterSystems Server view. For more information on this view, see InterSystems Server View.



When prompted, provide the following values:

- **Name of new server definition** — an arbitrary name to identify this server.
- **Description (optional)** — a brief description of the server.
- **Hostname or IP address of web server** — the host name or IP address of the web server that publishes the web services for your target InterSystems server through the InterSystems Web Gateway.
- **Port of web server** — the port number at which your web server hosts your target InterSystems server.
- **Path prefix of instance** (optional) — if you are connecting to your target InterSystems server using a web server that hosts multiple InterSystems servers, the URL component which precedes the application path to identify the target server.
- **Username** — the username VS Code should use when it authenticates to this server.
- **Confirm connection type** — the protocol used for connections. Possible values are `http` and `https`.

  Note: Additional configuration may be necessary if you want to establish an HTTPS connection using a self-signed certificate.

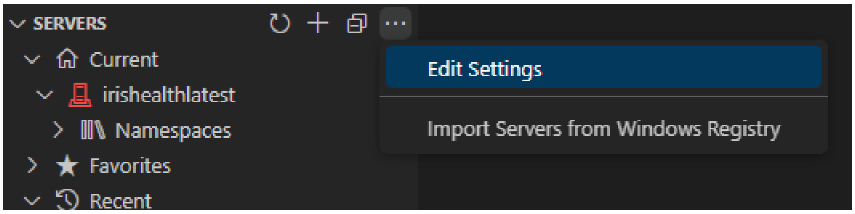
You can create a configuration for a server that is not currently running.

Once you have entered these values, the extension stores the server definition in your user-level settings.json file, and the server appears at the top of the **Recent** folder in the InterSystems Server view. See Editing a Server Configuration.

5.3.1 Editing a Server Configuration

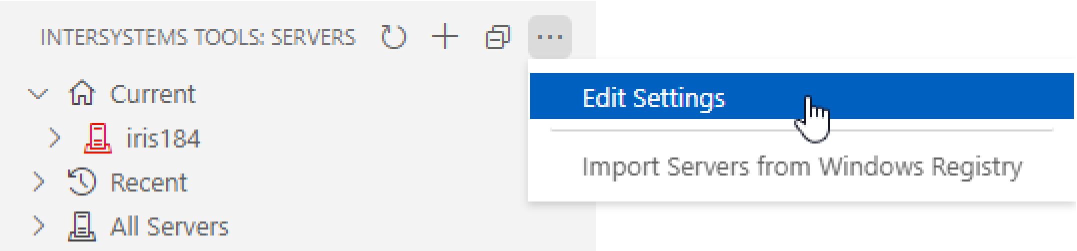
If you need to modify a server configuration, perform the following steps:

1. Select **File** > **Preferences** > **Settings** (**Code** > **Preferences** > **Settings** on Mac) from the menu.
2. Select the **User** settings level.
3. Select **Extensions** from the outline section of the editor pane.
4. Select InterSystems Server Manager from the list to find the InterSystems Server Manager area of the edit pane, as illustrated in the following screen shot:



5. Select **Edit in settings.json**.

You can also access the settings.json file within the InterSystems view container: select the **...** (ellipsis) button in the title bar of the **Servers** view, and then select **Edit Settings.**


INTERSYSTEMS TOOLS: SERVERS
Current
iris184
Recent
All Servers
Edit Settings
Import Servers from Windows Registry

A server configuration in the settings.json file has the following structure, populated with the values you entered when you configured the server:

```
{
    "intersystems.servers": {
        "iris-1": {
            "webServer": {
                "scheme": "http",
                "host": "localhost",
                "port": 443
            },
            "username": "_SYSTEM"
        }
    }
}
```

The components of this server definition are as follows:

- `"iris-1"` — the arbitrary name to identify the InterSystems IRIS server.
- `"webServer"` — the collection of properties which define the web server you use to connect to the InterSystems IRIS server through the InterSystems Web Gateway.
  - `"scheme"` — the protocol used for connections (`"http"` or `"https"`).

    Note: Additional configuration may be necessary if you want to establish an HTTPS connection using a self-signed certificate.

  - `"host"` — the host name or IP address for the web server.
  - `"port"` — the port number for the web server.
  - `"pathPrefix"` — where applicable, the directory path under which the web server publishes the target InterSystems IRIS server's web services.

- `"username"` — the username VS Code uses to authenticate to the server.
- `"password"` — password for the specified username. For security reasons, InterSystems strongly recommends authenticating through VS Code instead of specifying passwords as plaintext within the settings.json server definition.

5.3.2 Establish an HTTPS Connection Using a Self-Signed Certificate

Your organization may issue self-signed certificates to facilitate HTTPS connections to development or testing servers.

In order for VS Code to allow an HTTPS connection to an InterSystems server using a self-signed certificate, perform either of the following configuration changes:

- Within the VS Code User Settings, set variables as follows:
  - `"http.proxyStrictSSL": false`
  - `"http.proxySupport": "fallback"` or `"http.proxySupport": "off"`

  Then, save the configuration and reload VS Code.

  CAUTION: This configuration disables certificate checking, and is therefore inappropriate for use outside of a secure private network.

- Add the self-signed certificate to your operating system's root certificates. (Refer to the documentation for your operating system for specific instructions about how to do this.) Then, reload VS Code.

5.4 Configuring a Server Connection

To establish a connection between a server and a client-side workspace location, perform the following steps:

1. Open the folder where you want to store your client-side files.
2. Select the InterSystems view container button in the Activity Bar. This action opens a dialog that allows you to select an existing server to connect to or provide the information to configure a new server definition.
3. Follow the prompts to select a server and a namespace.

Once you have selected a server and namespace, connection configuration is complete. VS Code adds the server and namespace to the status bar.

You cannot create a connection to an InterSystems IRIS server if the InterSystems IRIS server is not running.

5.4.1 Editing a Server Connection

If you need to modify an existing server connection, perform the following steps:

1. Select **File** > **Preferences** > **Settings** (**Code** > **Preferences** > **Settings** on Mac) from the menu bar.
2. Select the **Workspace** settings level.
3. Search for `objectscript: conn`.
4. Select **Edit in settings.json**.

A connection configuration in the settings.json file has the following structure, populated with the values you specified when you configured the connection:

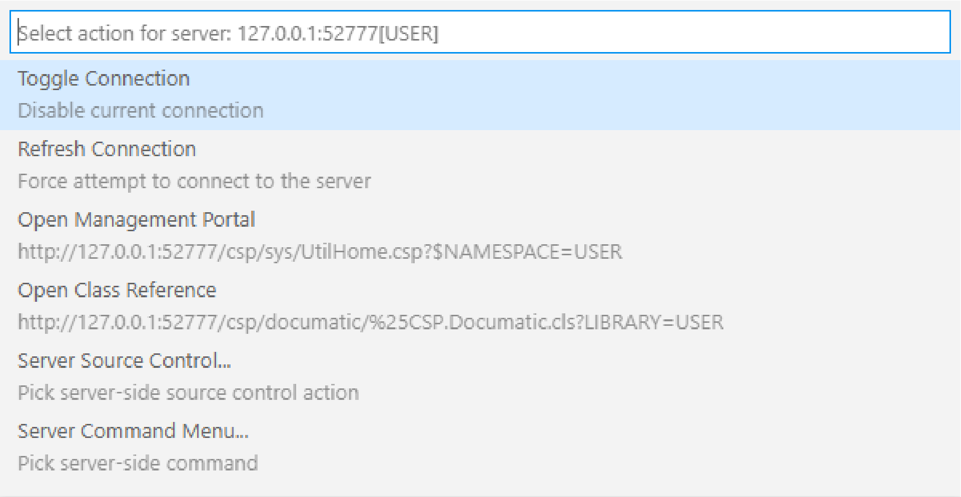
```
"objectscript.conn": {
  "ns": "USER",
  "server": "iris-1",
  "active": true,
},
```

The components of this configuration are as follows:

- `"ns"` — namespace to use on the server
- `"server"` — server name, as specified in the server configuration
- `"active"` — boolean value specifying whether the connection is active.

5.5 Add Custom Entries to the Server Actions Menu

Clicking on the server and namespace in the status bar opens a list of the actions you can take for this server, as illustrated in the following image:



Select action for server: 127.0.0.1:52777[USER]

Toggle Connection
Disable current connection

Refresh Connection
Force attempt to connect to the server

Open Management Portal
http://127.0.0.1:52777/csp/sys/UtilHome.csp?$NAMESPACE=USER

Open Class Reference
http://127.0.0.1:52777/csp/documatic/%25CSP.Documatic.cls?LIBRARY=USER

Server Source Control...
Pick server-side source control action

Server Command Menu...
Pick server-side command

You can add custom entries to this list using the `objectscript.conn.links` configuration object. This object consists of key-value pairs where the key is the label displayed in the menu and the value is the URI the action opens. The following variables are available for substitution in the URI:

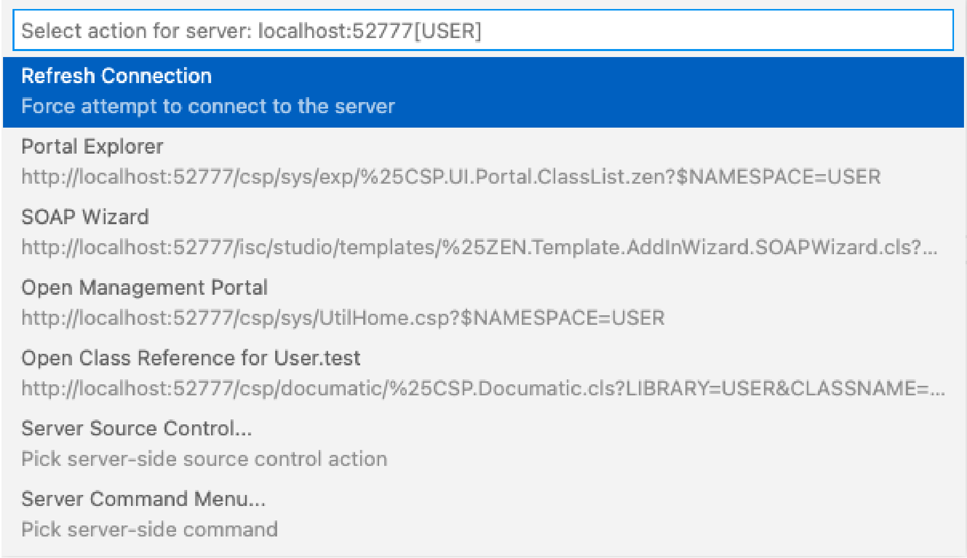
- `${host}` — the hostname of the connected server. For example: `localhost`
- `${port}` — the port of the connected server.
- `${serverURL}` — the full connection string for the server, including the protocol, <baseURL> for your instance, and path prefix. For example: `http://localhost:443/pathPrefix`
- `${ns}` — the namespace that you are connected to. The value is URL encoded. For example: `%25SYS` or `USER`

- `${namespace}` — The raw namespace parameter for the connection. For example, `user`
- `${classname}` — the name of the class which is currently open, or an empty string if the currently opened document is not a class.
- `${classnameEncoded}` — a URL encoded version of `${classname}`.
- `${project}` — the name of the server-side project which is currently open, or an empty string if no project is open.
- `${username}` — the username for the user account you are currently using to connect to the server.

Here is an example of a server action configuration:

```
"objectscript.conn": {
    "links": {
        "Portal Explorer": "${serverUrl}/csp/sys/exp/%25CSP.UI.Portal.ClassList.zen?$NAMESPACE=${ns}"
    },
}
```

The preceding server action configuration adds an entry to the Server Actions menu, as illustrated in the following image:



Select action for server: localhost:52777[USER]

Refresh Connection
Force attempt to connect to the server

Portal Explorer
http://localhost:52777/csp/sys/exp/%25CSP.UI.Portal.ClassList.zen?$NAMESPACE=USER

SOAP Wizard
http://localhost:52777/isc/studio/templates/%25ZEN.Template.AddInWizard.SOAPWizard.cls?...

Open Management Portal
http://localhost:52777/csp/sys/UtilHome.csp?$NAMESPACE=USER

Open Class Reference for User.test
http://localhost:52777/csp/documatic/%25CSP.Documatic.cls?LIBRARY=USER&CLASSNAME=...

Server Source Control...
Pick server-side source control action

Server Command Menu...
Pick server-side command

6

Develop ObjectScript Unit Tests

The InterSystems® ObjectScript extension for Visual Studio Code (VS Code) leverages VS Code's testing API and the InterSystems %UnitTest framework to enable you to run and debug methods from classes which extend %UnitTest.TestCase, right within the VS Code interface.

Regardless of whether you are editing on the client-side or the server-side, you can accelerate development for your ObjectScript project by integrating unit test development into your VS Code workflow.

6.1 Prerequisites

Running or debugging ObjectScript tests within a VS Code workspace requires the following:

- Visual Studio Code version 1.83.0 or later.
- InterSystems ObjectScript extension for VS Code version 2.12.1 or later.
- An active server connection to an instance of InterSystems IRIS version 2023.3 or later.
  - The value of the ^UnitTestRoot global for the instance must be set to an existing directory within the local filesystem. (This is required so that the extension can execute unit tests using %UnitTest.Manager methods. The extension does not use the value of ^UnitTestRoot.)

6.2 Setup

To run or debug ObjectScript tests within VS Code:

1. Open the workspace that you are developing tests in. Tests can be run from a single- or multi-root workspace, client- or server-side. (See Configure Your Workspace.)

   Note: You cannot run unit tests within the `%SYS` namespace of an InterSystems server.

2. For client-side workspace folders: edit the ObjectScript extension setting `objectscript.unitTest.relativeTestRoots` so that it specifies the relative paths within your workspace folder where your test classes are located. See the User and Workspace Settings section of the VS Code documentation for further guidance with editing settings.
3. If test cases within a test directory require additional resources:

- For *client-side* workspace folders: configure and populate an autoload subdirectory, as described in Autoload Client-Side Resources.
- For *server-side* workspace folders: ensure that all required resources are already loaded on the server.

6.2.1 How the Extension Locates Your Unit Tests

Once you have configured your workspace as described in the preceding steps, the ObjectScript extension locates classes which extend %UnitTest.TestCase and makes them available for running and debugging within the VS Code interface.

- For a client-side workspace, the extension locates applicable test classes within the directories specified by `objectscript.unitTest.relativeTestRoots`.
- For a server-side workspace, the extension locates applicable test classes within all workspace folders which are configured to include them. For example:
  - a server-side workspace folder which includes the contents of a project will display tests if the project includes applicable test classes.
  - a server-side workspace folder which includes the contents of the filter `filter=*.mac` will not display tests even if there are test classes in the namespace.

Within the test directory, the extension ignores test cases which are located in any directory with a name beginning with an underscore character (_) except in such cases where the name of the autoload directory begins with an underscore. This is consistent with the behavior of the %UnitTest.Manager class's RunTest() method.

Note: If you are editing on the client-side and you export a test class from the server using the ObjectScript Explorer, the extension places that class within a /src/ folder on the client-side workspace, regardless of where the test is stored on the workspace. This may result in duplicate copies of a class in two client-side locations.

When using a client-side editing workflow, InterSystems recommends editing on the client exclusively, and importing changes from the client onto the server.

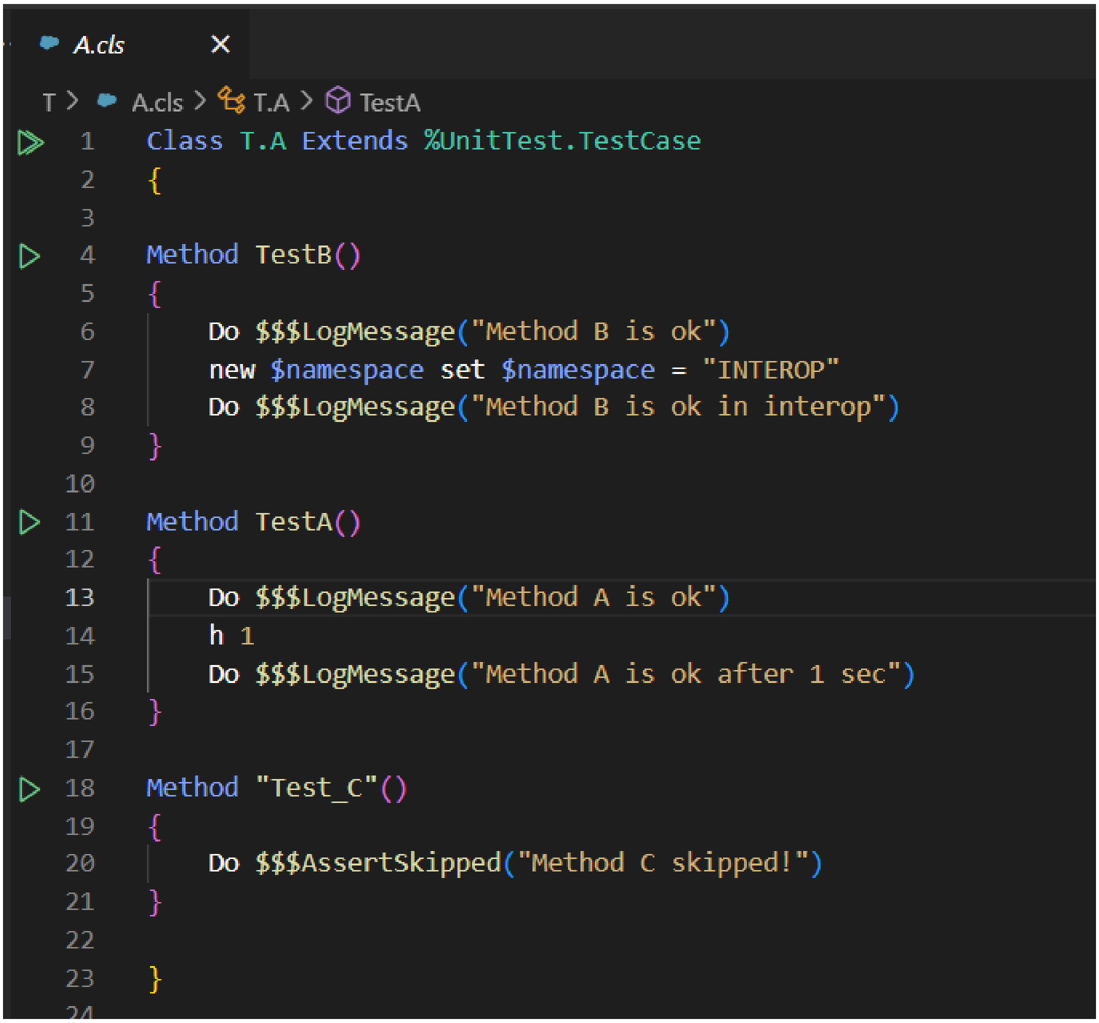
6.2.2 Autoload Client-Side Resources

When you initiate the running or debugging of a test method in a client-side workspace, the ObjectScript extension can automatically load resources which your test methods require from a subdirectory within the test directory. These resources are loaded into a temporary location onto the server before the test case classes; upon conclusion of the running or debugging session, the extension deletes both the test case classes and the autoload resources from the server.

By default, the extension is configured to autoload UDL and XML files from /_autoload/ if a directory with that name exists. You can modify the name of the autoload directory and control which types of files the extension loads within the InterSystems ObjectScript extension settings.

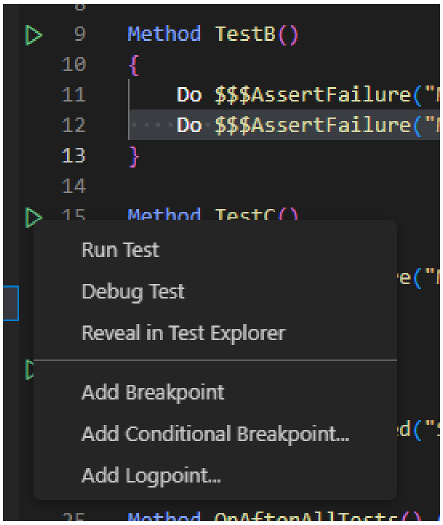
6.3 Run and Debug Your Unit Tests

For a test case class which is open in an editor, you can run a test case by selecting the green play button ▷ which appears in as a margin decoration at the start of the method definition (beside the line number). To run all the test methods for the class, select the "play all" button ▷ at the beginning of the class definition. These margin decorations are visible in the following image:



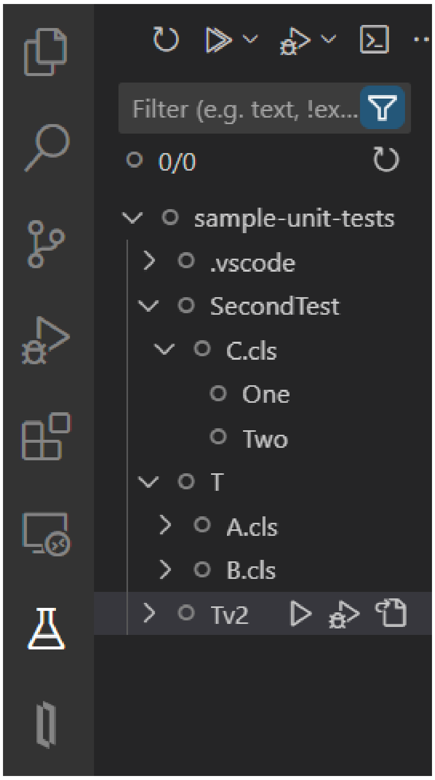
```
Class T.A Extends %UnitTest.TestCase
{

Method TestB()
{
    Do $$$LogMessage("Method B is ok")
    new $namespace set $namespace = "INTEROP"
    Do $$$LogMessage("Method B is ok in interop")
}

Method TestA()
{
    Do $$$LogMessage("Method A is ok")
    h 1
    Do $$$LogMessage("Method A is ok after 1 sec")
}

Method "Test_C"()
{
    Do $$$AssertSkipped("Method C skipped!")
}

}
```

As shown in the following image, additional options are available by right-clicking one of these play buttons—including the option to debug:


```
Method TestB()
{
    Do $$$AssertFailure("
    Do $$$AssertFailure("
}

Method TestC()
```

See Running and Debugging for further guidance on debugging ObjectScript methods within VS Code.

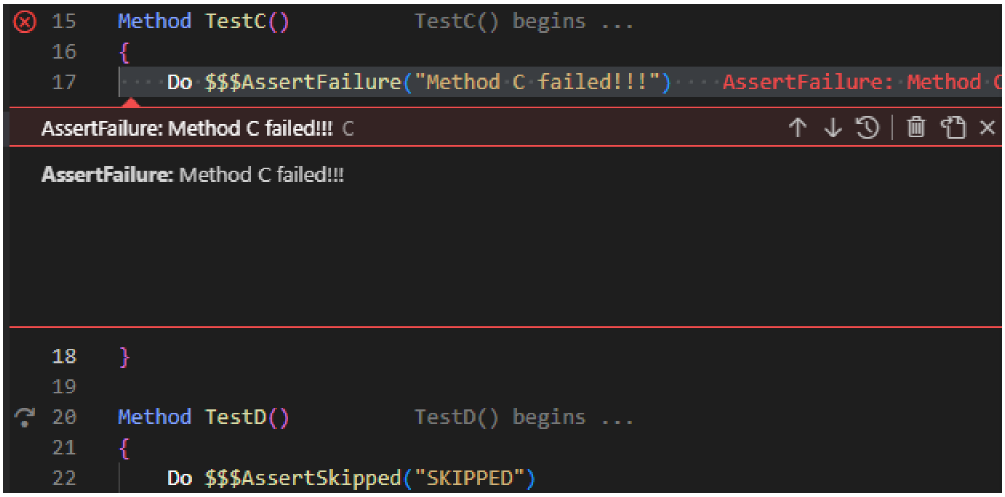
You can also run and debug tests within the *Testing Explorer*, which is available by selecting the beaker icon from the Activity Bar. The Testing Explorer presents all the test methods in your workspace in a tree view, as shown in the following image:



In addition to the graphic interfaces, the ObjectScript extension makes a variety of testing-related commands available in the Command Palette. To review a list of these commands, activate the Command Palette and search for “`Test:`”

6.3.1 View Test Results

After running or debugging a test case, the decoration in the margin of the editor updates to reflect the state of the test result. The editor also allows you to view the output of an individual test inline. Both of these features are visible in the following image:



```
Method TestC()          TestC() begins ...
{
    Do $$$AssertFailure("Method C failed!!!")    AssertFailure: Method
```

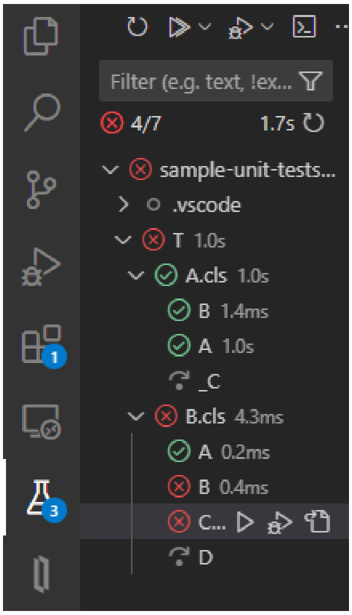
AssertFailure: Method C failed!!! C

AssertFailure: Method C failed!!!

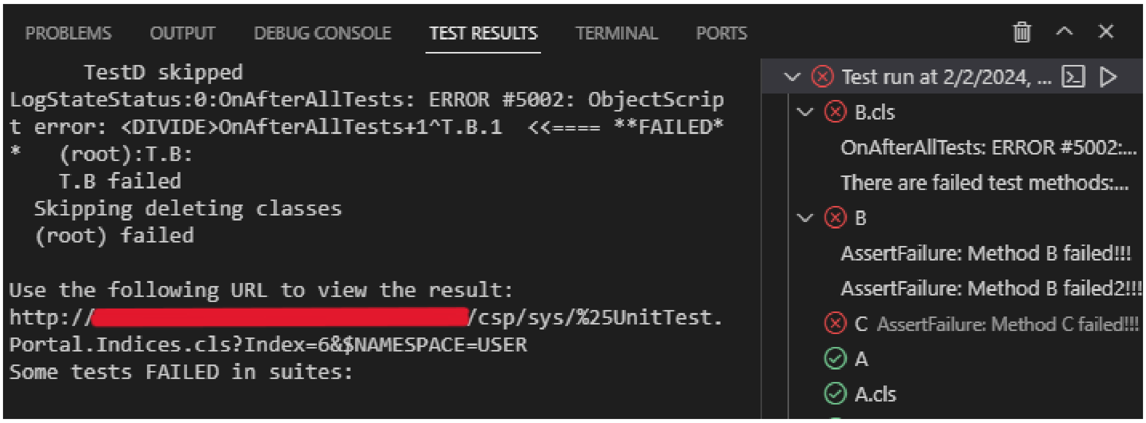
```
}

Method TestD()          TestD() begins ...
{
    Do $$$AssertSkipped("SKIPPED")
```

To view test output within the editor in this way, select the **Peek Output** or **Peek Error** option from the context menu for the test's margin decoration, or navigate to the desired test result from the inline output view for another test using the arrow buttons.

After running or debugging a test case, the Testing Explorer also updates to reflect the state of the test result, as depicted in the following image:



You can also view results for all the tests you've run in the **Test Results** panel, depicted in the image which follows. A side bar provides a collapsible outline of results for any test runs you have not cleared; selecting a result from the sidebar displays a detailed view of results in the primary section of the panel.


PROBLEMS
OUTPUT
DEBUG CONSOLE
TEST RESULTS
TERMINAL
PORTS
Test run at 2/2/2024, ...
B.cls
OnAfterAllTests: ERROR #5002:...
There are failed test methods:...
B
AssertFailure: Method B failed!!!
AssertFailure: Method B failed2!!!
C AssertFailure: Method C failed!!!
A
A.cls

```
      TestD skipped
LogStateStatus:0:OnAfterAllTests: ERROR #5002: ObjectScrip
t error: <DIVIDE>OnAfterAllTests+1^T.B.1  <<==== **FAILED*
*   (root):T.B:
    T.B failed
  Skipping deleting classes
  (root) failed

Use the following URL to view the result:
http://[redacted]/csp/sys/%25UnitTest.
Portal.Indices.cls?Index=6&$NAMESPACE=USER
Some tests FAILED in suites:
```

You can clear test results by selecting that option from the **Test Result** panel's menu bar, the editor's inline test result view, and the Test Explorer sidebar's menu bar.

7

Running and Debugging

The InterSystems ObjectScript Extension for Visual Studio Code (VS Code) provides support for ObjectScript debugging. It takes advantage of the debugging capabilities built into VS Code, so you may find these VS Code documentation resources useful:

- Debugging Intro Video
- Debugging User Guide

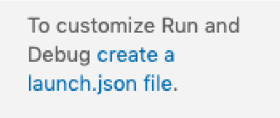
InterSystems also produced a short video which walks through the steps in this documentation page.

7.1 Debug Configurations

In order to run or debug an ObjectScript class or routine or attach to a running process, you must create a debug configuration. Some other languages default to running the currently active file, but to run ObjectScript, you must specify the routine or ClassMethod to use or the running process to attach to.

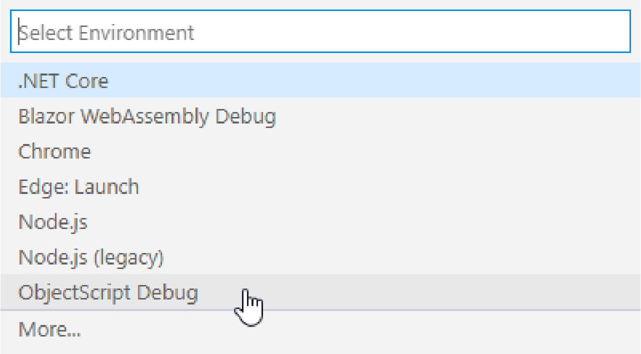
Select the **Run and Debug** button in the Activity Bar:

If no debug configurations are available, you are prompted to create one, as illustrated in the following image:



To customize Run and Debug create a launch.json file.

Clicking the link opens a dialog containing a list of debug environments, as illustrated in the following image. Select *ObjectScript Debug* from the list.



Once you have chosen a debug environment, VS Code creates and opens a launch.json file which contains the following default content:

```
{
    "version": "0.2.0",
    "configurations": [
        {
            "type": "objectscript",
            "request": "launch",
            "name": "XDebug"
        }
    ]
}
```

These attributes are mandatory for any debug configuration:

- `"type"` — identifies the type of debugger to use. In this case the value, `"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 optional attributes:

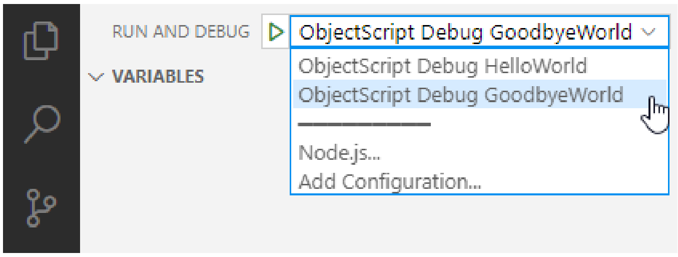
- `“processId”` — specifies the ID of process to attach to as a string or number. Defaults to `"${command:PickProcess}"`, which provides a drop-down list of process ID’s to attach to at runtime.
- `“system”` — specifies whether to allow attaching to system process. Defaults to `false`.

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
    },
  ]
}
```

7.2 Starting a Debugging Session

You can select a debug configuration from the list VS Code provides in the **Run and Debug** field, available in the title bar of the debug view:


RUN AND DEBUG
ObjectScript Debug GoodbyeWorld
VARIABLES
ObjectScript Debug HelloWorld
ObjectScript Debug GoodbyeWorld
Node.js...
Add Configuration...

Selecting the green arrow runs the selected debug configuration.

When you start an ObjectScript `"launch"` debug session, make sure that the file containing the program that you are debugging is open in your editor and is the active tab. VS Code will start a debug session with the server of the file in the active editor (the tab that the user is focused on). This also applies to ObjectScript `“attach”` debug 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 that the InterSystems server’s web server allows WebSocket connections.

Debugging commands and items on the **Run** menu function much as they do for other languages supported by VS Code. For information on VS Code debugging, see the documentation resources listed at the start of this section.

7.3 Debugging a REST Service

The InterSystems ObjectScript Extension provides a Webview-based graphical user interface that allows you to send a REST request and automatically start debugging the process on the server that handles it. With the InterSystems file that you want to debug open in the active text editor, you can show the GUI using the **Debug REST Service...** command. The command can be accessed in the Command Palette, the editor context menu, or the editor tab context menu. Follow the directions in the GUI to build your REST request and click the **Start Debugging** button to send the request and connect the debugger. Be sure you have a breakpoint set somewhere in the code that handles the request.

7.4 Troubleshooting Debugger Issues

If you are experiencing issues using the debugger, please follow these steps before opening an issue on GitHub. Note that the trace global may contain confidential information, so you should review the contents and mask/remove anything that you want to keep private.

1. Open the Terminal for the InterSystems IRIS instance you are connecting to, ensuring you are in the namespace which contains the class or routine you are debugging.
2. Run the command `Kill ^IRIS.Temp.Atelier("debug")`, then `Set ^IRIS.Temp.Atelier("debug") = 1`. These commands turn on the Atelier API debug logging feature. If you are on Caché or Ensemble, the global to modify in these commands `^CacheTemp.ISC.Atelier("debug")`.
3. In VS Code, start a debugging session using the configuration that produces the error.
4. When the error appears, copy the contents of the `^IRIS.Temp.Atelier("debug")` global. Include this global in your GitHub issue.
5. After you capture the log, run the command `Kill ^IRIS.Temp.Atelier("debug")`, then `Set ^IRIS.Temp.Atelier("debug") = 0`. This turns logging off again.

7.5 Using the WebSocket Terminal

The InterSystems ObjectScript Extension provides support for a WebSocket-based command-line interface for executing ObjectScript commands on a connected server. The server can be on the same system as VS Code, or a remote system. This feature is only supported when connecting to InterSystems IRIS version 2023.2 or later.

The WebSocket terminal supports the following features:

- VS Code's shell integration feature so your command history and output will be captured by VS Code and can be accessed by its UI.
- Multi-line editing. An additional editable line will be added when the user presses **Enter** and there are unclosed `{` or `(` in the command input.
- Syntax coloring for command input. (You can toggle this using the `objectscript.webSocketTerminal.syntaxColoring` setting)
- Syntax checking for entered command input, with detailed error messages reported along with the standard `<SYNTAX>` error.
- Many features of the standard terminal, including:
  - The Read command
  - Interrupts (**Ctrl-C**)
  - Namespace switches
  - Custom terminal prompts (except code 7)
  - Shells such as SQL (`Do $SYSTEM.SQL.Shell()`) and Python (`Do $SYSTEM.Python.Shell()`)

The WebSocket terminal does not support command-line debugging since the InterSystems ObjectScript Extension contains an interactive debugger. Users are also discouraged from using routine editing commands since VS Code with the InterSystems ObjectScript Extension Pack provides an excellent ObjectScript editing experience.

Note that the terminal process starts using the **JOB** command, so if you have a ^%ZSTART routine enabled the `JOB` subroutine will be called at the start of the process, not `LOGIN` like the standard terminal. Also, the ZWELCOME routine will not run before the first command prompt is shown.

The WebSocket terminal can be opened from the command palette using the **ObjectScript: Launch WebSocket Terminal** command. The WebSocket terminal connection is established using the current server connection. A WebSocket terminal connection can also be opened from the Terminal Profiles menu.

7.5.1 Troubleshooting WebSocket Terminal Issues

If you are experiencing issues using the WebSocket Terminal, please follow these steps before opening an issue on GitHub. Note that the trace global may contain confidential information, so you should review the contents and mask/remove anything that you want to keep private.

1. If you are using a Microsoft Internet Information Services (IIS) web server to connect, confirm that the IIS **WebSocket Protocol** feature is enabled.
2. Open the Terminal for the InterSystems IRIS instance you are connecting to, ensuring you are in the namespace which contains the class or routine you are debugging.
3. Run the command `Kill ^IRIS.Temp.Atelier("terminal")`, then `Set ^IRIS.Temp.Atelier("terminal") = 1`. These commands turn on the Atelier API debug logging feature.
4. In VS Code, launch the WebSocket terminal and run the commands that produce the error.
5. When the error appears, copy the contents of the `^IRIS.Temp.Atelier("terminal")` global. Include this global in your GitHub issue.
6. After you capture the log, run the command `Kill ^IRIS.Temp.Atelier("terminal")`, then `Set ^IRIS.Temp.Atelier("terminal") = 0`. This turns logging off again.

8

Server-Side Editing

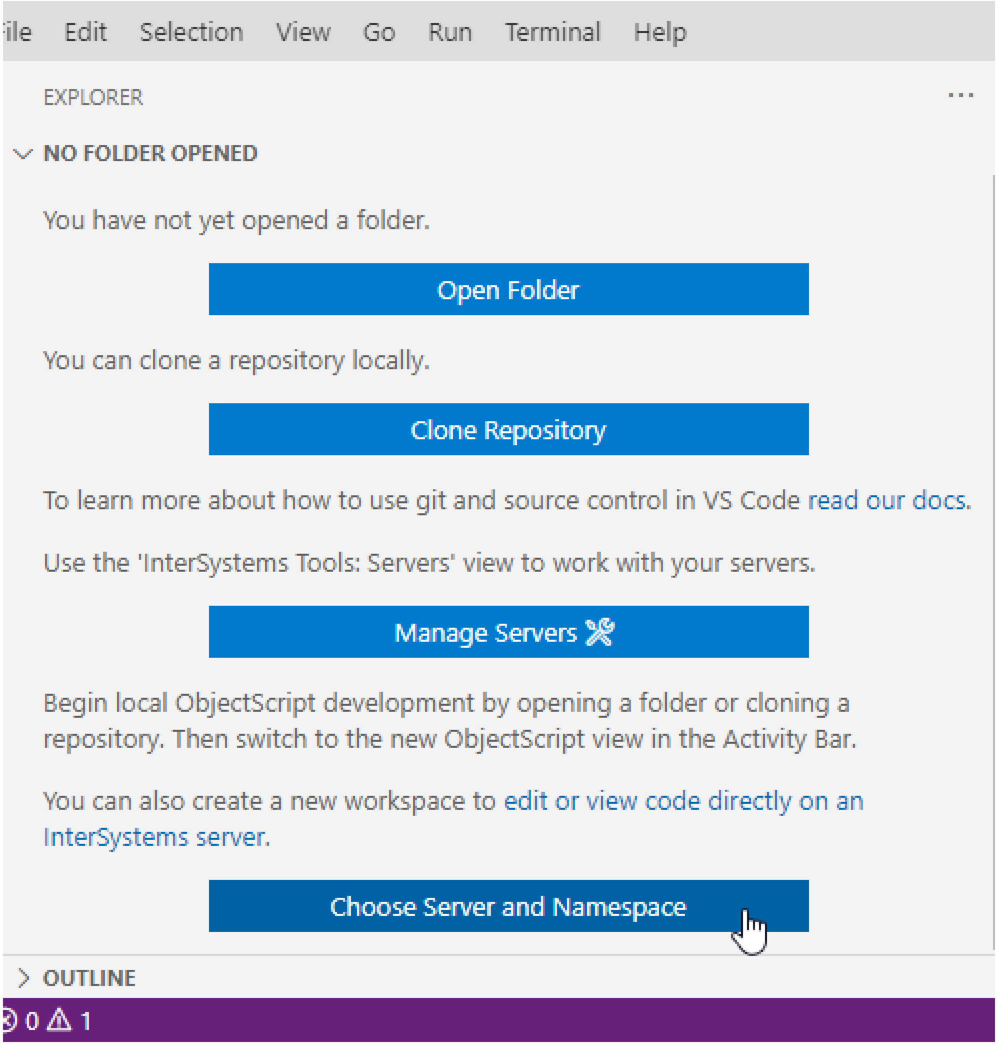
You can configure the InterSystems ObjectScript extension to edit code directly on the server, using the multi-root workspaces feature in Visual Studio Code (VS Code). This type of configuration is useful in cases where source code is stored in a Source Code Management (SCM) product interfaced to the server. For example you might already be using the Source Control menu in InterSystems Studio or the Management Portal, implemented by a source control class that extends %Studio.SourceControl.Base.

8.1 Configuring for Server-Side Editing

First configure the `intersystems.servers` entry for your server, as described in Configuring a Server.

Next, create a workspace for editing code directly on the server by performing the following steps:

1. Open VS Code. You must perform the following steps starting with no folder or workspace open, so if a folder or workspace is already open, close it.
2. Open the Explorer view, if it is not already visible.
3. Click the button labeled **Choose Server and Namespace** in the Explorer view, as shown in the image below:



You have not yet opened a folder.

You can clone a repository locally.

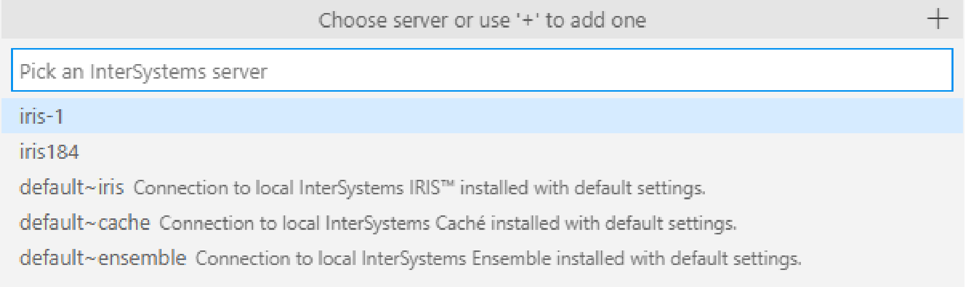
To learn more about how to use git and source control in VS Code read our docs.

Use the 'InterSystems Tools: Servers' view to work with your servers.

Begin local ObjectScript development by opening a folder or cloning a repository. Then switch to the new ObjectScript view in the Activity Bar.

You can also create a new workspace to edit or view code directly on an InterSystems server.

4. Pick the name of an existing server configuration from the list, or click the + (plus) button to create a new server configuration.



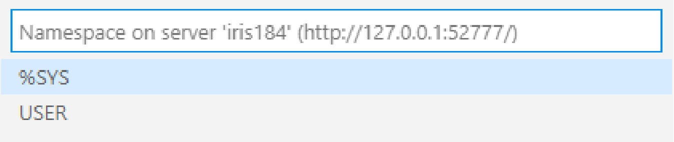
Choose server or use '+' to add one

Pick an InterSystems server

- iris-1
- iris184
- default~iris Connection to local InterSystems IRIS™ installed with default settings.
- default~cache Connection to local InterSystems Caché installed with default settings.
- default~ensemble Connection to local InterSystems Ensemble installed with default settings.

5. Enter login credentials, if prompted.

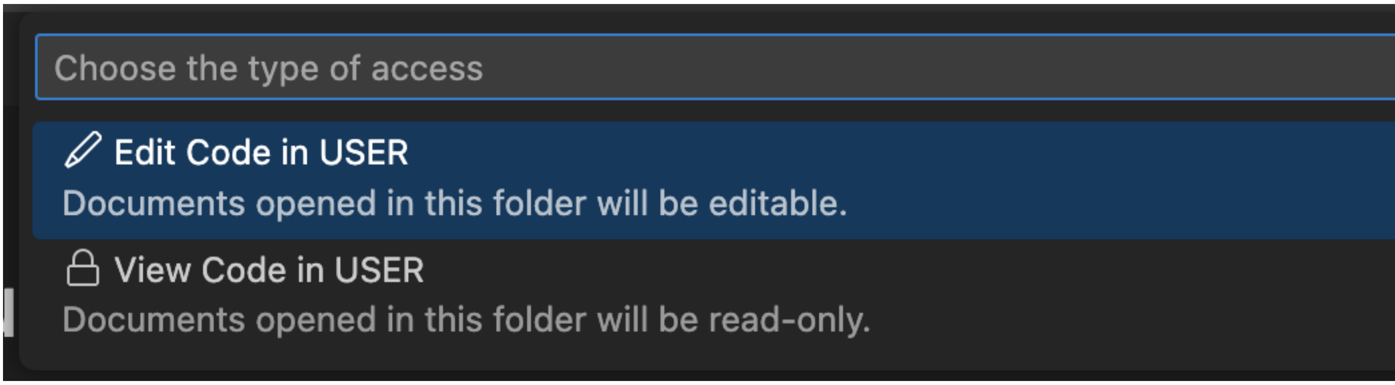
6. Choose a namespace from the list retrieved from the target server.



Namespace on server 'iris184' (http://127.0.0.1:52777/)

- %SYS
- USER

7. Choose an access mode (editable or read-only) from the list.

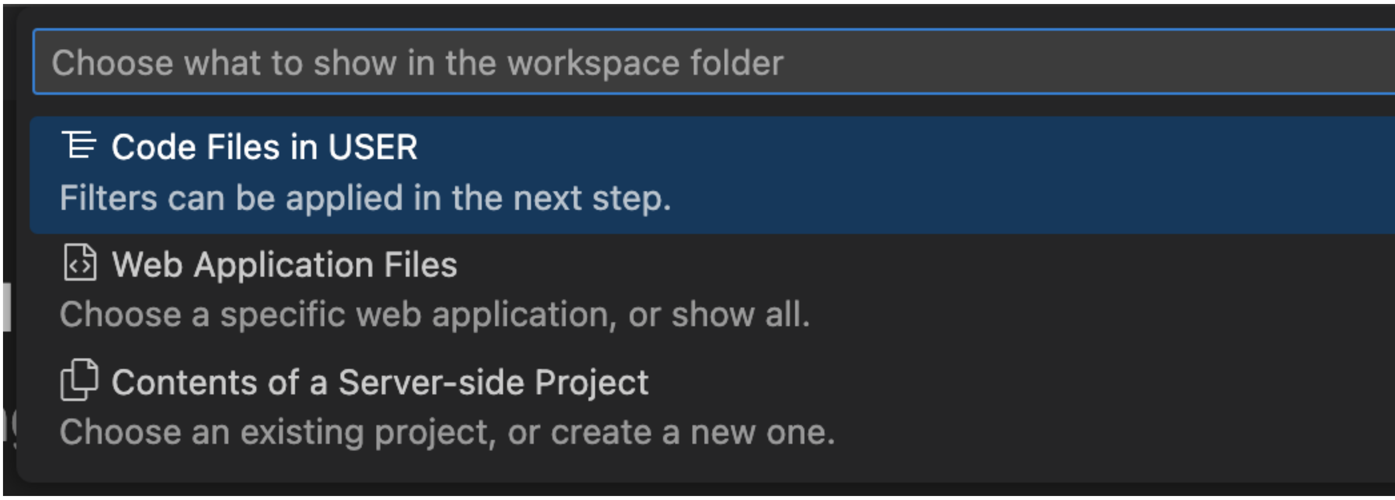


Choose the type of access

Edit Code in USER
Documents opened in this folder will be editable.

View Code in USER
Documents opened in this folder will be read-only.

8. Choose the category of files to display (code files, web application files, or contents of a server-side project).



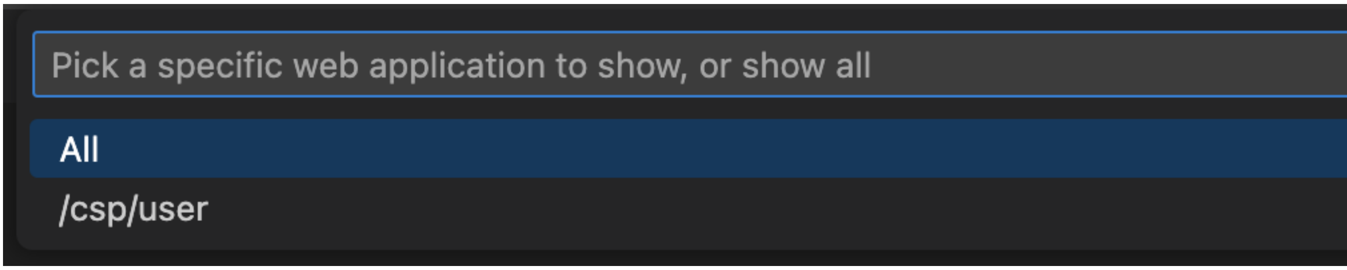
Choose what to show in the workspace folder

Code Files in USER
Filters can be applied in the next step.

Web Application Files
Choose a specific web application, or show all.

Contents of a Server-side Project
Choose an existing project, or create a new one.

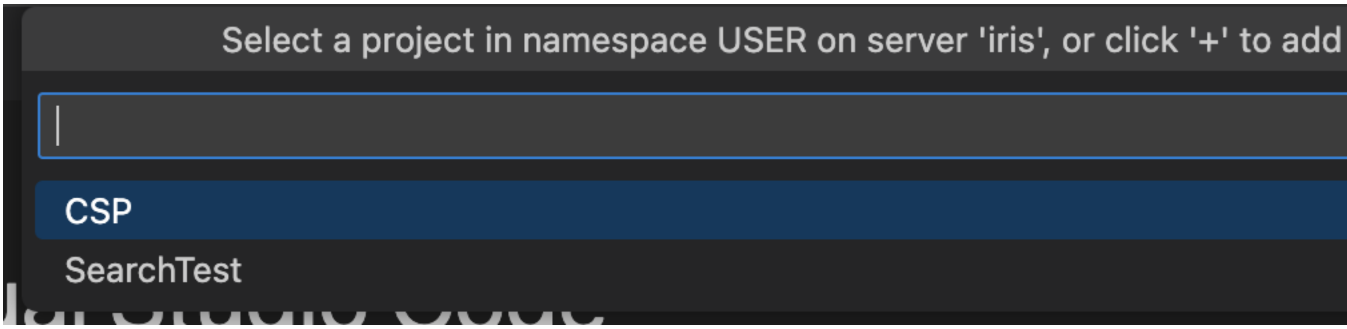
- If you chose to show web application files, choose an optional web application to show files from:



Pick a specific web application to show, or show all

All
/csp/user

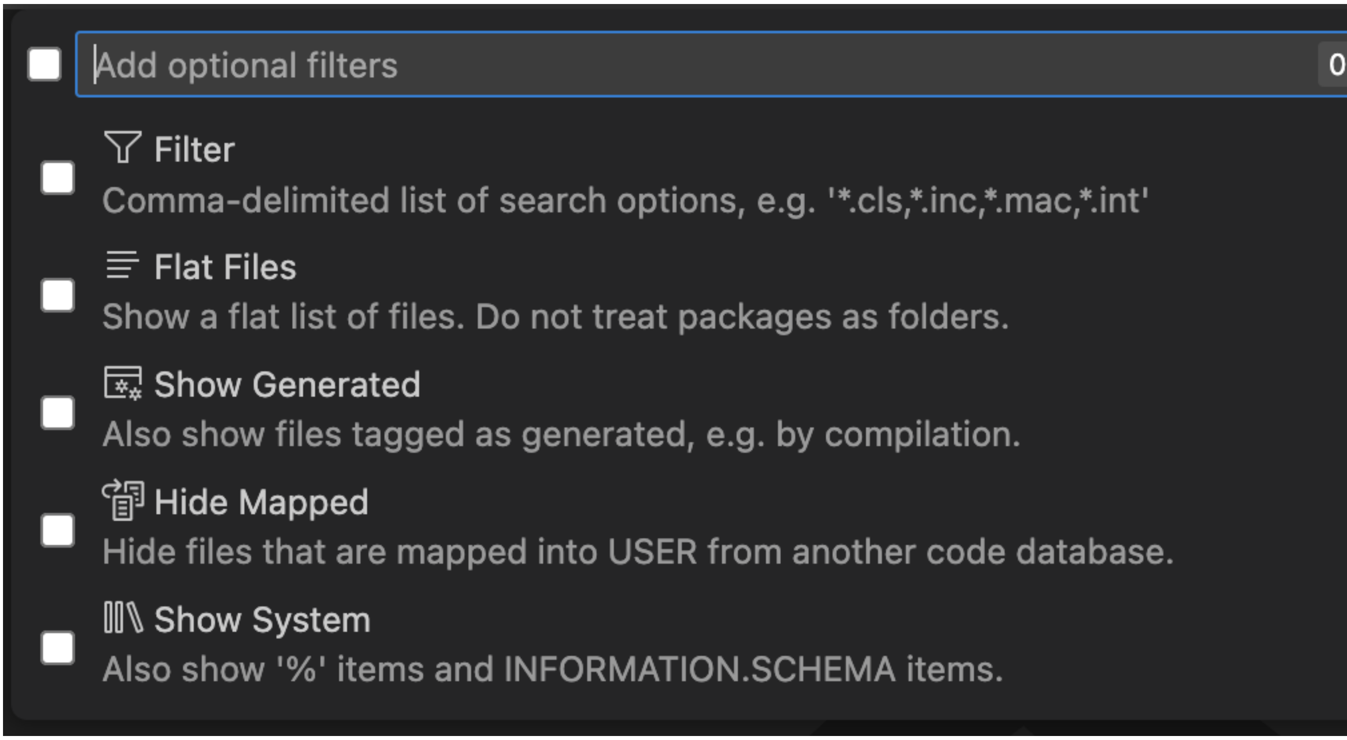
- If you choose to show a project's contents, choose the project:



Select a project in namespace USER on server 'iris', or click '+' to add

CSP
SearchTest

- If you create your own filter of files to display, pick the filter options:



☐ Add optional filters

☐ **Filter**
Comma-delimited list of search options, e.g. '*.cls,*.inc,*.mac,*.int'

☐ **Flat Files**
Show a flat list of files. Do not treat packages as folders.

☐ **Show Generated**
Also show files tagged as generated, e.g. by compilation.

☐ **Hide Mapped**
Hide files that are mapped into USER from another code database.

☐ **Show System**
Also show '%' items and INFORMATION.SCHEMA items.

9. If you want to reopen this workspace in the future, select **File** > **Save Workspace As...** to save the details of this workspace as a .code-workspace file.

Note that the ObjectScript Explorer view is not visible in the InterSystems view container. Because the files listed in the Explorer view are all on the server, the ObjectScript Explorer is not needed for this configuration.

The .code-workspace file is a JSON file which you can edit directly, as described in Workspaces. What follows is a simple example:

```
{
  "folders": [
    {
      "name": "iris184:USER",
      "uri": "isfs://iris184:user"
    }
  ],
  "settings": {}
}
```

- The `"name"` property provides a name for this server-side folder.
- The `"uri"` property indicates the location of resources on the server. The supplied value has three components:
  - The first component can be either `isfs` or `isfs-readonly`. These values specify that the folder is on an InterSystems IRIS server. `isfs-readonly` specified read-only access.
  - The value following the `/` (forward slash) specifies the name of the server.
  - The value following the `:` (colon) specifies the namespace in lowercase characters.

The string `isfs` which appears in the `"uri"` for folders configured for server-side editing is an abbreviation created by InterSystems which stands for InterSystems File Service. It implements the VS Code FileSystemProvider API, which lets you make any remote location look like a local one. It works well for making artefacts in an InterSystems IRIS namespace look like local files.

To add more root folders to your workspace, giving you access to code in a different namespace, or on a different server, use the context menu on your existing root folder to invoke the **Add Server Namespace to Workspace...** command. This command is also available in the Command Palette.

The example which follows describes a two-folder workspace in which the second folder gives read-only access to the %SYS namespace:

```
{
  "folders": [
    {
      "name": "iris184:USER",
      "uri": "isfs://iris184:user"
    },
    {
      "name": "iris184:%SYS (read-only)",
      "uri": "isfs-readonly://iris184:%sys"
    }
  ],
  "settings": {}
}
```

Workspaces can also consist of a mixture of server-side folders and local folders. Use the context menu's **Add Folder to Workspace...** option to add a local folder.

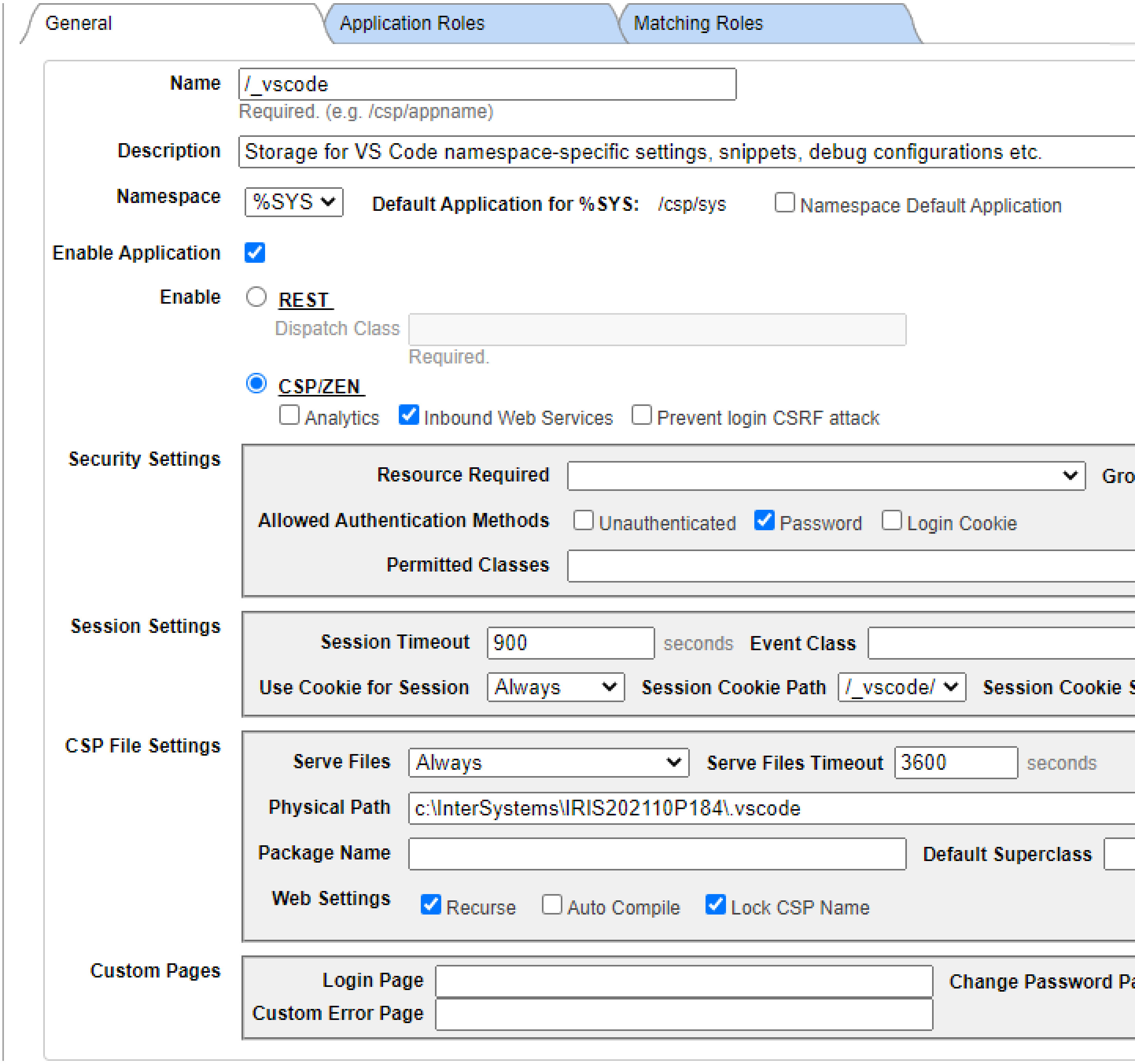
Root folders can be re-sequenced using drag/drop in the Explorer view, or by editing the order their definition objects appear within the `"folders"` array in the JSON.

8.2 Configuring Storage for Folder-Specific Settings

When you use VS Code to edit source code on the client, the settings model allows you to specify folder-specific settings in a .vscode\settings.json file located in a workspace root folder. These settings take precedence when you work under that workspace root folder.

If you use an isfs-type workspace to operate directly in a namespace on a server, you need to configure that server to support storing and serving up the .vscode\settings.json file. The .vscode subfolder of a workspace root folder also stores folder-specific code snippets and debug configurations. These are available when using this configuration.

Use the Management Portal for your InterSystems IRIS instance to create a web application named `_vscode` on the server. Select **System Administration** > **Security** > **Applications** > **Web Applications**, then **Create New Web Application**:



Name /_vscode
Required. (e.g. /csp/appname)

Description Storage for VS Code namespace-specific settings, snippets, debug configurations etc.

Namespace %SYS **Default Application for %SYS:** /csp/sys ☐ Namespace Default Application

Enable Application ☑

Enable ○ REST
Dispatch Class
Required.
● CSP/ZEN
☐ Analytics ☑ Inbound Web Services ☐ Prevent login CSRF attack

Security Settings
Resource Required **Gro**
Allowed Authentication Methods ☐ Unauthenticated ☑ Password ☐ Login Cookie
Permitted Classes

Session Settings
Session Timeout 900 seconds **Event Class**
Use Cookie for Session Always **Session Cookie Path** /_vscode/ **Session Cookie S**

CSP File Settings
Serve Files Always **Serve Files Timeout** 3600 seconds
Physical Path c:\InterSystems\IRIS202110P184\.vscode
Package Name **Default Superclass**
Web Settings ☑ Recurse ☐ Auto Compile ☑ Lock CSP Name

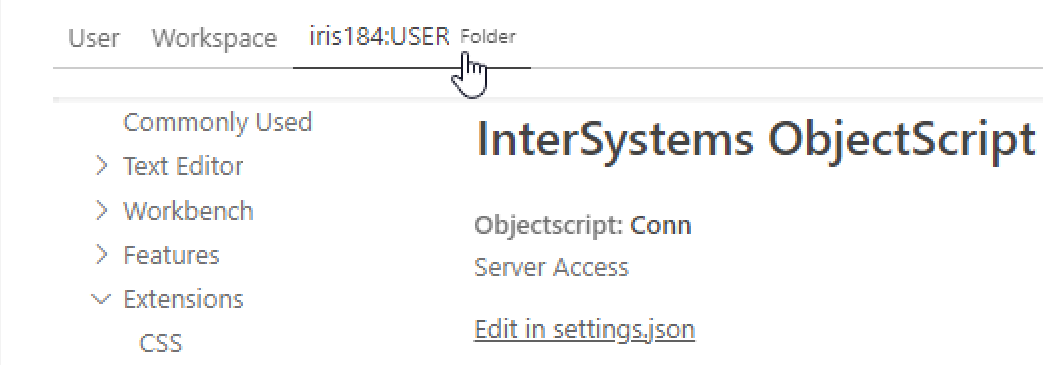
Custom Pages
Login Page **Change Password Pa**
Custom Error Page

Enter the following values:

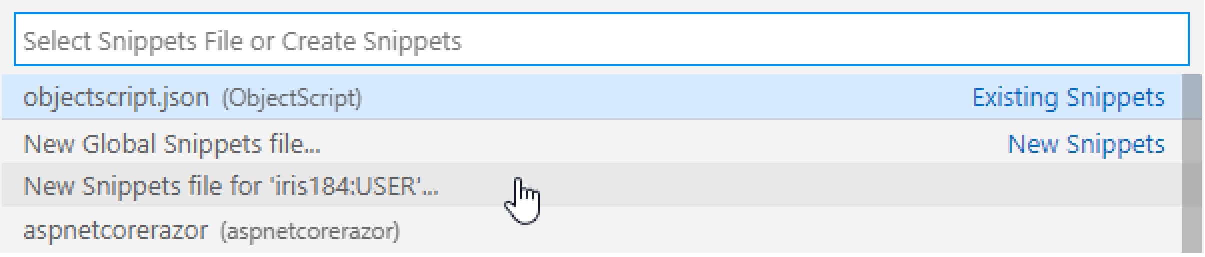
- **Name:** —enter `/_vscode`
- **Description** — a brief description
- **Namespace** — select %SYS
- **Enable Application** — select
- **Enable** — select **CSP/ZEN**
- **Allowed Authentication Methods** — select **Password**
- **CSP File Settings: Physical Path** — enter a physical path appropriate for your platform and your installation folder

- **CSP File Settings: Web Settings** — clear the **Auto Compile** option

Be sure to save the configuration. If you have an isfs-type workspace root folder that connects to a namespace on this server as a user with the **`%DB_IRISSYS:READ`** privilege, you can now write and read folder-specific settings:



You can also create a folder-specific snippets file by selecting the **Preferences: Configure User Snippets** command:


Select Snippets File or Create Snippets
objectscript.json (ObjectScript)
Existing Snippets
New Global Snippets file...
New Snippets
New Snippets file for 'iris184:USER'...
aspnetcorerazor (aspnetcorerazor)

To edit the server-side namespace-specific files for all namespaces directly through VS Code, add an isfs-type root folder with the following URI:

```
isfs://servername:%sys/_vscode?csp
```

For a single namespace (for example, USER) the URI would be as follows:

```
isfs://servername:%sys/_vscode/USER?csp
```

8.3 Web Application (CSP) Files

To edit web application files (also known as CSP files) on a server, configure the URI as follows:

```
isfs://myserver:xxx{csp_application}?csp
```

For example, the following URI gives you access to the server-side files of the /csp/user application. The `csp` query parameter is mandatory and the suffix on the server name must specify the correct namespace for the web application. For example:

```
"uri": "isfs://myserver:user/csp/user?csp"
```

Changes you make to files opened from this root folder of your VS Code workspace will be saved onto the server.

8.4 Filters and Display Options

The query string of the `"uri"` property accepts several parameters that control filtering and display of the server-side entities. The examples below access the USER namespace on the server whose definition is named ‘myserver’.

- `isfs://myserver:user/?generated=1` shows generated files as well as not generated files
- `isfs://myserver:user/?project=prjname` shows only files in project `prjname`. Cannot be combined with any other parameter.
- `isfs://myserver:user/?mapped=0` hides files that are mapped from a non-default database
- `isfs://myserver:user/?filter=%Z*.mac,%z*.mac` — a comma-delimited list of search options, ignoring type. The default is *.cls, *.inc, *.mac, *.int. To see all files, use `*`.
- `isfs://myserver:user/Utils?filter=*.cls` shows only .cls files within the Utils package (or packages with names which begin with the string `Utils`).

  Note: As suggested by the preceding example, the wildcard character (`*`) cannot be used in a medial position within a filter string (for example, to filter by both an initial package name string and a file type). In other words, `isfs://myserver:user/?filter=Utils*.cls` would filter for files with the literal name `Utils*.cls`.

The options `generated` and `mapped` can be combined with each other, and with `filter`.

To modify the query parameters or name of an existing workspace folder, run the **Modify Server-Side Workspace Folder...** command from the Command Palette or the Explorer context menu.

8.5 Advanced Workspace Configurations

This section gives examples of some more complex workspace definitions for server-side editing.

Use **File** > **New File** to create a new file. Add content similar to the following example. Note that `“my-project”` in the `isfs://` URI, should be the same as the `“name”` property (that is, the root display name) of any local folder where specialized settings for the connection are being stored in a .vscode/settings.json file.

```
{
  "folders": [
    {
      "name": "my-project",
      "path": ".",
    },
    {
      "uri": "isfs://my-project",
      "name": "server"
    }
  ],
  "settings": {
  }
}
```

Save the file, giving it an arbitrary name with the extension .code-workspace. VS Code shows you a button with an offer to open this workspace. Select the button.

The next time VS Code starts, two folders appear in the root directory with the names described in the .code-workspace file. Expand the server folder to see code on the configured server and namespace, routines and classes in one place. You can now edit this code. If you have SourceControl class, it should be configured the way, to export files in the same location which used in VS Code workspace.

The following is an example which implements a connection to multiple namespaces on the same InterSystems IRIS server:

```
{
  "folders": [
    {
      "name": "myapp",
      "path": ".",
    },
    {
      "uri": "isfs://myapp",
      "name": "server",
    },
    {
      "uri": "isfs://myapp:user",
      "name": "user",
    },
    {
      "uri": "isfs://myapp:%sys",
      "name": "system",
    },
    {
      "uri": "isfs://user@hostname:port?ns=%SYS",
      "name": "system (alternative syntax)",
    }
  ],
  "settings": {
    "files.exclude": {},
    "objectscript.conn": {
      "active": true,
      "username": "_system",
      "password": "SYS",
      "ns": "MYAPP",
      "port": 52773,
    }
  }
}
```

9

Work with Projects

A project is a named set of class definitions, routines, include files, web application files or custom documents. All files in a project must be in the same namespace on the same InterSystems server. Each document can be associated with any number of projects. Each namespace can contain any number of projects.

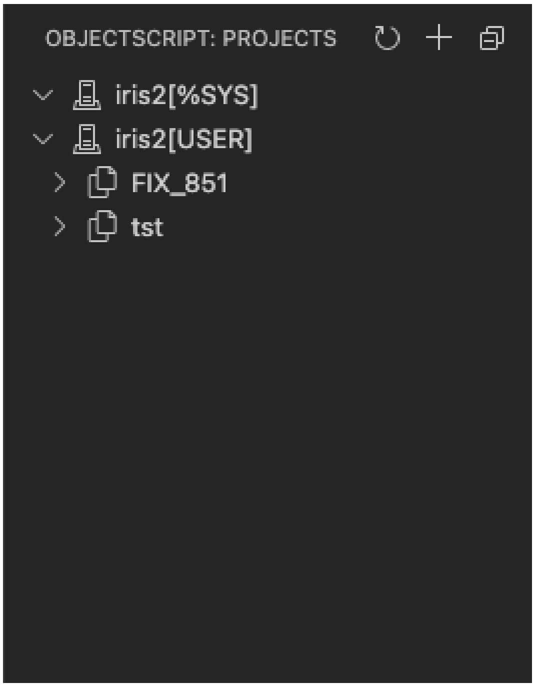
9.1 Why Projects

You are not required to use projects in VS Code, but you should consider using them if:

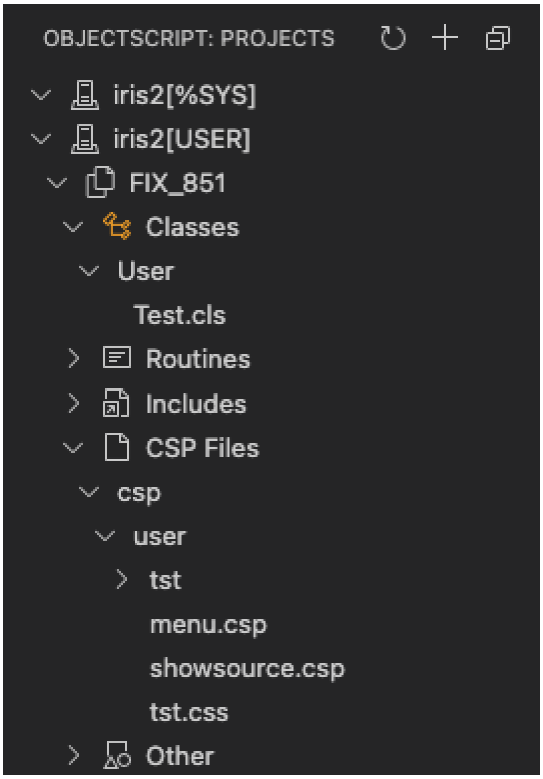
- You work server-side and the type and filter query parameters are not granular enough.
- You work server-side and want to edit CSP and non-CSP files in the same workspace folder.
- You work client-side and want to group together many files to export with a single click.
- You are migrating from InterSystems Studio and want to keep using an existing project.

9.2 InterSystems Projects Explorer

The easiest way to manage projects is using the Projects Explorer, which is in the InterSystems view container:



Initally, the Projects explorer contains a root node for each server and namespace connection that exists for the current workspace. It can be expanded to show all projects in that namespace on the server, and expanding the project node will show its contents:



You can also add root nodes for namespaces on any server configured using the InterSystems Server Manager extension. To do so, select the + (plus) button in the title bar of the view.

The following sections will describe how to use the Projects Explorer and other tools to work with projects.

9.3 Creating Projects

There are two ways to create projects in VS Code:

- Right-click on a server-namespace node in the Projects Explorer and select the **Create Project** menu option.
- Open the Command Palette and select the **ObjectScript: Create Project** command.

Project names are required to be unique per server-namespace and may optionally have a description. The description is shown when hovering over the project’s node in the Projects Explorer or below its name when selecting one in a drop-down menu.

9.4 Modifying Projects

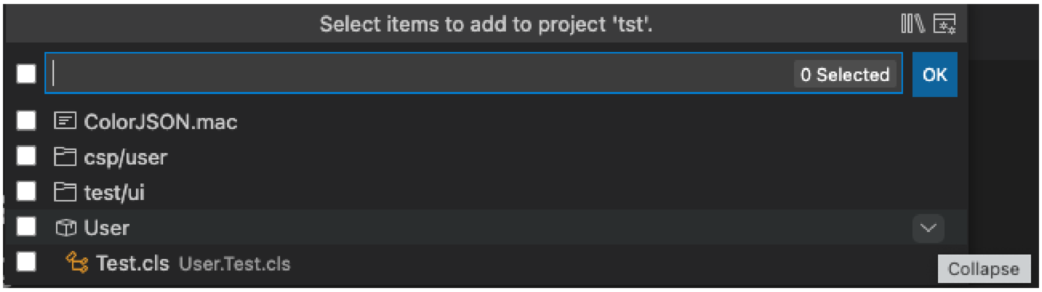
There are three ways to add or remove items from a project:

- Using the Projects Explorer:
  - To add items, right-click on the project node or one of the document type nodes (i.e. Classes or Routines) and select the **Add Items to Project...** menu option. If you selected on a document type node, you will only be shown documents of that type to add.
  - To remove an item, right-click on its node and select the **Remove from Project** menu option. If you remove a package or directory node, all of its children will also be removed from the project. You may also right-click on the project node and select the **Remove Items from Project...** menu option to be presented with a multi-select dropdown that allows you to remove multiple items at once.
- Within a workspace folder configured to view or edit documents in a project directly on the server:
  - To add items, right-click a root `isfs[-readonly]` folder that has the `project` query parameter in its URI and select the **Add Items to Project...** menu option.
  - To remove an item, right-click on its node and select the **Remove from Project** menu option. If you remove a package or directory node, all of its children will also be removed from the project. You may also right-click on a root `isfs[-readonly]` folder that has the `project` query parameter in its URI and select the **Remove Items from Project...** menu option to be presented with a multi-select dropdown that allows you to remove multiple items at once.
- Using commands:

  Open the Command Palette and select the **ObjectScript: Add Items to Project...** or **ObjectScript: Remove Items from Project...** command.

9.4.1 Add to Project UI

The **Add to Project** command implements a custom multi-select dropdown that is shown regardless of how it is invoked. Items that are in the namespace and not already in the project are shown. The rest of this section describes the elements of this UI:



- Title bar row:
  - Select the icons to show or hide system and generated items, respectively.
- Input box row:
  - Select the check box to select all items that are currently shown.
  - Type in the input box to filter the items that are shown.
  - Select the **OK** button to add the selected items to the project.
- Item rows:
  - Select the check box to select the item. If the item is a package or CSP directory, all of its contents will be selected as well, even though the check boxes for those items don’t appear selected.
  - The icon preceding the name represents its type. It corresponds to the icons in the Projects Explorer and ObjectScript Explorer.
  - The more prominent text is the short name of the item, as it would appear in a file system.
  - The less prominent text is the full name of the item, including its package or CSP directory.
  - Select the arrow icon for each item to show or hide its contents.

9.5 Deleting Projects

There are two ways to delete projects in VS Code:

- Right-click on a project node in the Projects Explorer and select the **Delete Project menu** option.
- Open the Command Palette and run the **ObjectScript: Delete Project** command.

9.6 Editing Project Contents Server-Side

There are a few methods to create a workspace folder to view or edit documents in a project directly on the server:

- Follow the steps here and select the project.
- Right-click in the Explorer view and select the **Add Server Namespace to Workspace** menu option.
- Right-click on a project node in the Projects Explorer and select the **Add Workspace Folder For Project** menu option.

- Add a folder to your .code-workspace file directly:

  ```
  {
    "uri": "isfs://myserver:user/?project=prjname",
    "name": "prjname"
  }
  ```

9.7 Editing Project Contents Client-Side

The entire contents of the project can be easily exported to your local file system for client-side editing. To do so, simply right-click on the project you'd like to export and select the **Export Project Contents** menu option.

9.8 Notes

If you are getting `ERROR #5540: SQLCODE: -99 Message: User abc is not privileged for the operation` when you try to expand the Projects Explorer or view a project's contents in a virtual folder, then grant user abc (or a SQL role they hold) the following SQL permissions:

```
GRANT SELECT, INSERT, UPDATE, DELETE ON %Studio.Project, %Studio.ProjectItem TO abc
GRANT EXECUTE ON %Studio.Project_ProjectItemsList TO abc
```

10

Import and Export InterSystems Documents as XML Files

Using the InterSystems ObjectScript Extensions for VS Code, you can export source code documents from an InterSystems server into an XML file. You can also import source code documents from such an XML file onto an InterSystems server. This feature provides a convenient way to back up a large set of files spanning multiple packages, or to transfer such a set of files between InterSystems servers.

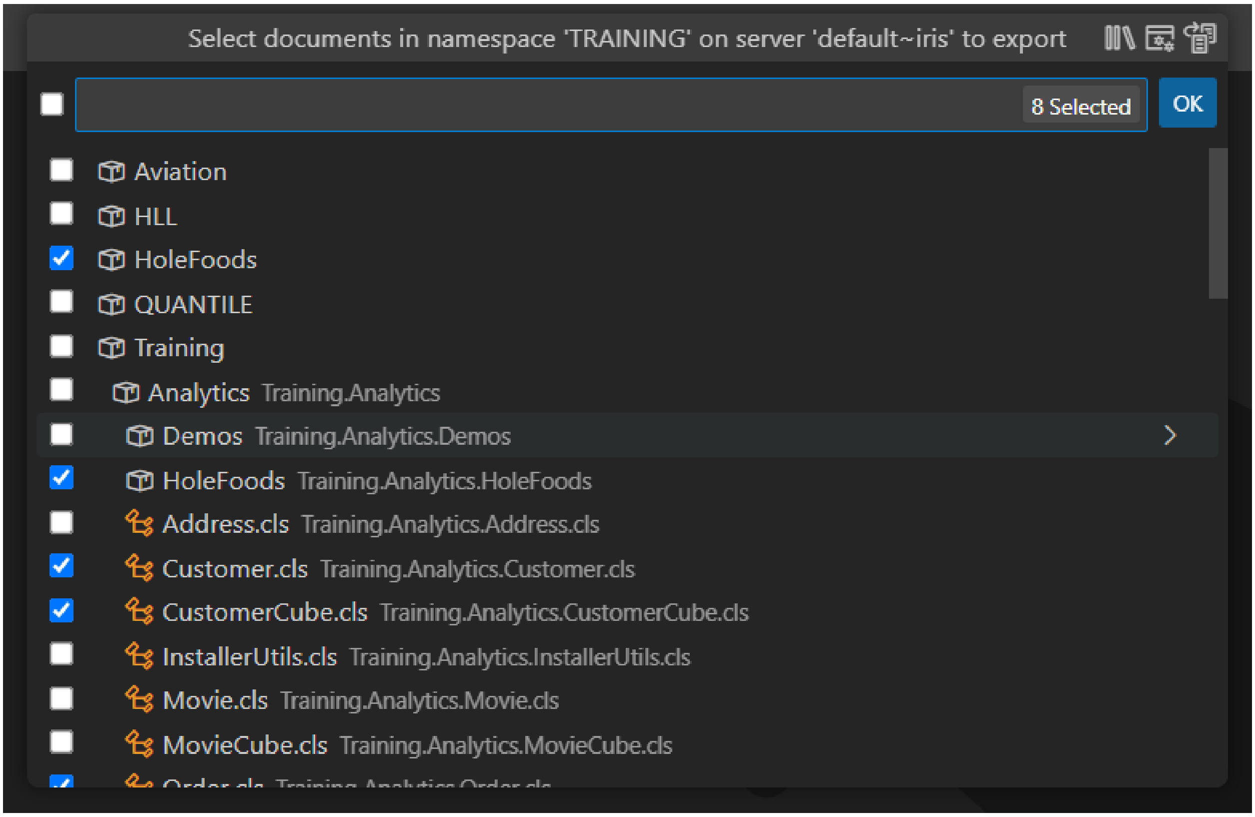
You can use this feature from any saved workspace with an active server connection.

Note: Currently, this feature is exclusively compatible with InterSystems IRIS and InterSystems IRIS for Health versions 2023.2 and up.

10.1 Export Documents as an XML File

To export multiple source code documents from an InterSystems server namespace into a single XML file on your local file system, perform the following steps:

1. From a saved workspace with an active server connection, open the Command Palette and run the **ObjectScript: Export Documents to XML File...** command.
2. If your workspace consists of multiple folders, use the menu to select the workspace folder connected to the server location from which you want to export documents.
3. From the menu, select the checkboxes that correspond to the documents you want to export. You can select entire packages, or expand the menu item for a package to select among the documents it contains:



Select documents in namespace 'TRAINING' on server 'default~iris' to export

☐ 8 Selected OK

☐ Aviation
☐ HLL
☑ HoleFoods
☐ QUANTILE
☐ Training
☐ Analytics Training.Analytics
☐ Demos Training.Analytics.Demos
☑ HoleFoods Training.Analytics.HoleFoods
☐ Address.cls Training.Analytics.Address.cls
☑ Customer.cls Training.Analytics.Customer.cls
☑ CustomerCube.cls Training.Analytics.CustomerCube.cls
☐ InstallerUtils.cls Training.Analytics.InstallerUtils.cls
☐ Movie.cls Training.Analytics.Movie.cls
☐ MovieCube.cls Training.Analytics.MovieCube.cls
☑ Order.cls Training.Analytics.Order.cls

4. Within the **Save As** window, navigate to the file system location where you want to save the XML file and specify a **File name**.
5. Select **Export**.

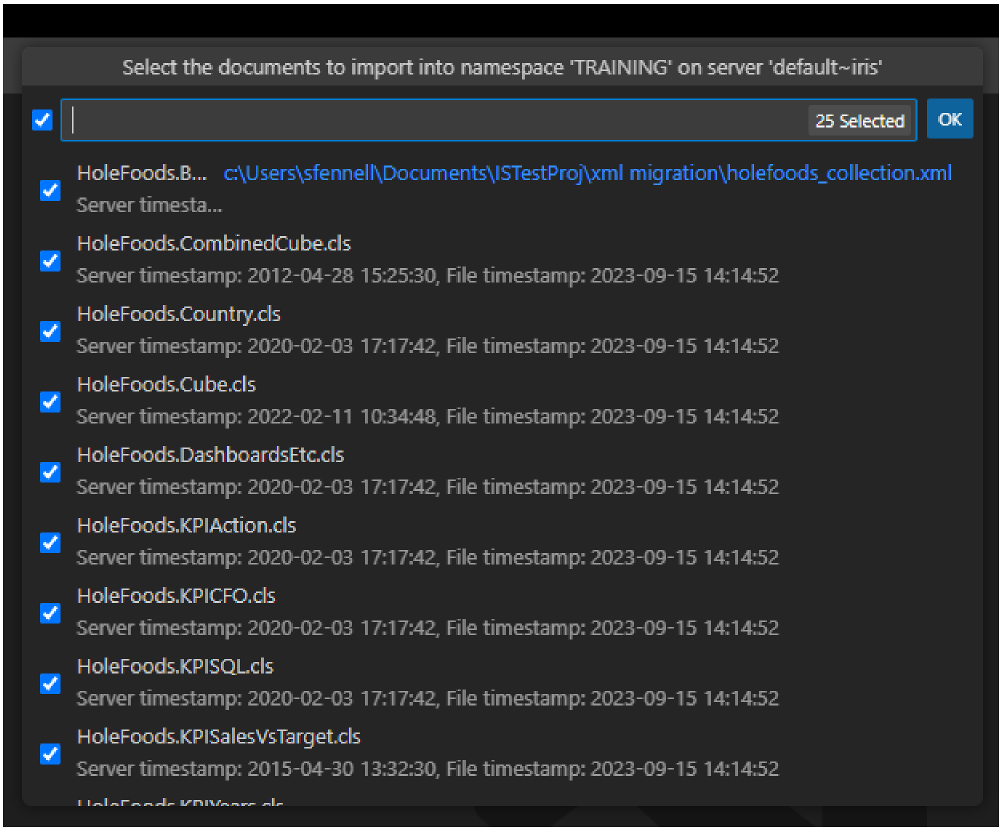
Functionally, this produces the same result as executing the %SYSTEM.OBJ.Export() method on the target server with the appropriate arguments.

10.2 Import Documents from an XML File

To import the contents of one or more XML export files from a local file system location into an InterSystems server namespace, perform the following steps:

Important: If the target namespace contains a source code document with the same name as a document recorded within the XML export file you are importing, the import operation replaces the existing file with the version of the document which the XML file's version of the document.

1. From a saved workspace with an active server connection, open the Command Palette and run the **ObjectScript: Import XML Files...** command.
2. If your workspace consists of multiple folders, select the workspace folder connected to the server location to which you want to import documents.
3. Within the **Open** window, locate and select the XML file (or files) from which you want to import documents.
4. Using the menu, confirm the documents you want to import:



Select the documents to import into namespace 'TRAINING' on server 'default~iris'

☑ 25 Selected OK

☑ HoleFoods.B... c:\Users\sfennell\Documents\ISTestProj\xml migration\holefoods_collection.xml
Server timesta...

☑ HoleFoods.CombinedCube.cls
Server timestamp: 2012-04-28 15:25:30, File timestamp: 2023-09-15 14:14:52

☑ HoleFoods.Country.cls
Server timestamp: 2020-02-03 17:17:42, File timestamp: 2023-09-15 14:14:52

☑ HoleFoods.Cube.cls
Server timestamp: 2022-02-11 10:34:48, File timestamp: 2023-09-15 14:14:52

☑ HoleFoods.DashboardsEtc.cls
Server timestamp: 2020-02-03 17:17:42, File timestamp: 2023-09-15 14:14:52

☑ HoleFoods.KPIAction.cls
Server timestamp: 2020-02-03 17:17:42, File timestamp: 2023-09-15 14:14:52

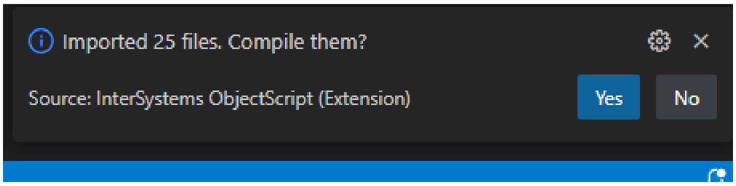
☑ HoleFoods.KPICFO.cls
Server timestamp: 2020-02-03 17:17:42, File timestamp: 2023-09-15 14:14:52

☑ HoleFoods.KPISQL.cls
Server timestamp: 2020-02-03 17:17:42, File timestamp: 2023-09-15 14:14:52

☑ HoleFoods.KPISalesVsTarget.cls
Server timestamp: 2015-04-30 13:32:30, File timestamp: 2023-09-15 14:14:52

Clear the checkboxes that correspond to documents which you do not want to import, as necessary.

5. Select **OK**.
6. Once the documents have been imported, a VS Code notification provides you the option to compile them:



Imported 25 files. Compile them?

Source: InterSystems ObjectScript (Extension)

Select **Yes** to compile the documents. Select **No** to decline.

Functionally, this produces the same result as executing the %SYSTEM.OBJ.Load() method on the target server with the appropriate arguments.

11

Low-Code Editors

VS Code contains support for low-code editors via its Custom Editors API. As InterSystems redevelops its suite of low-code editors for Interoperability components, support for integration with this extension will be included. This page lists the currently supported low-code editors and describes how to use them in VS Code.

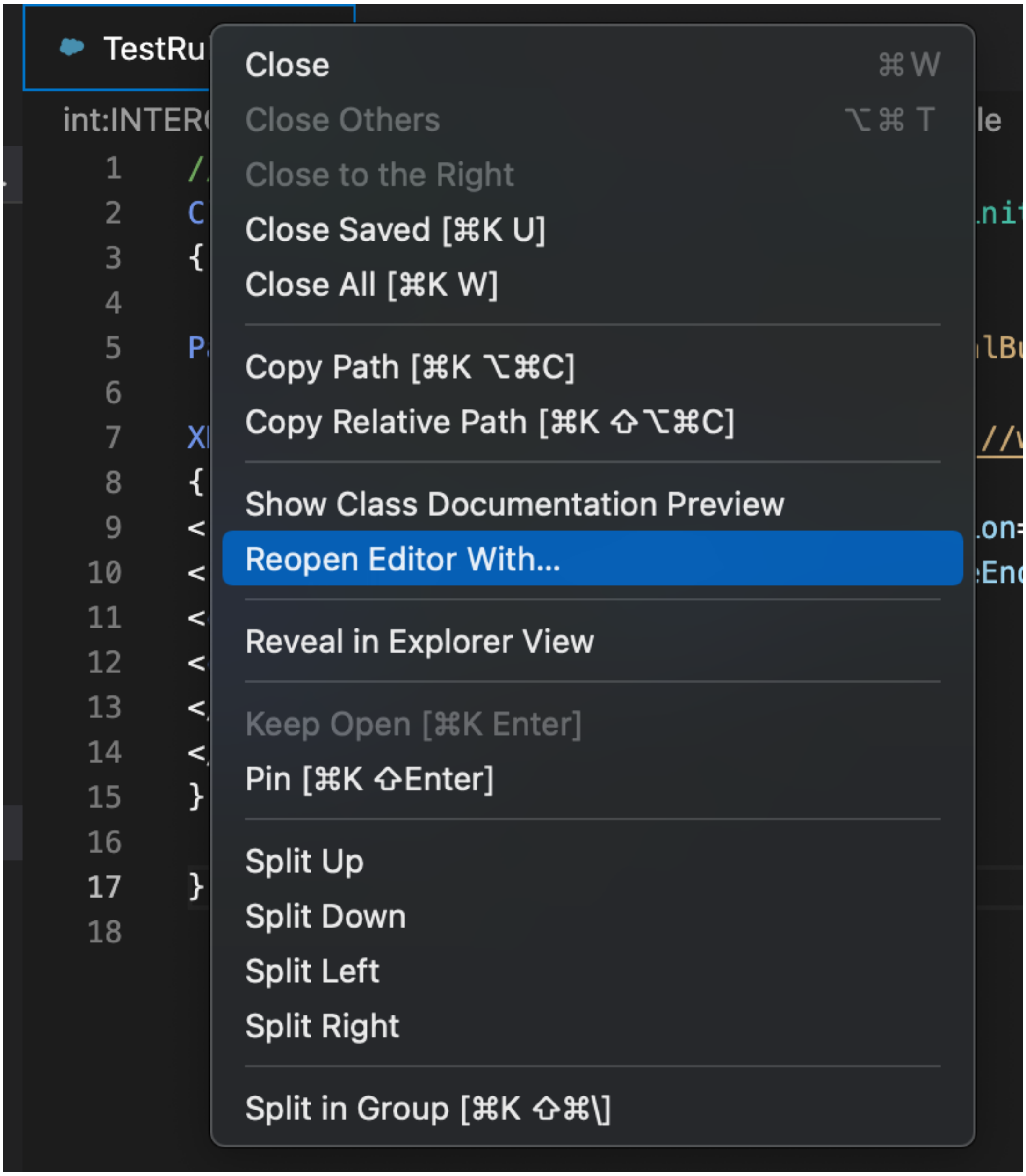
11.1 Supported Editors

The following list contains all InterSystems low-code editors that support integration with VS Code, along with the earliest version of InterSystems IRIS that contains the support:

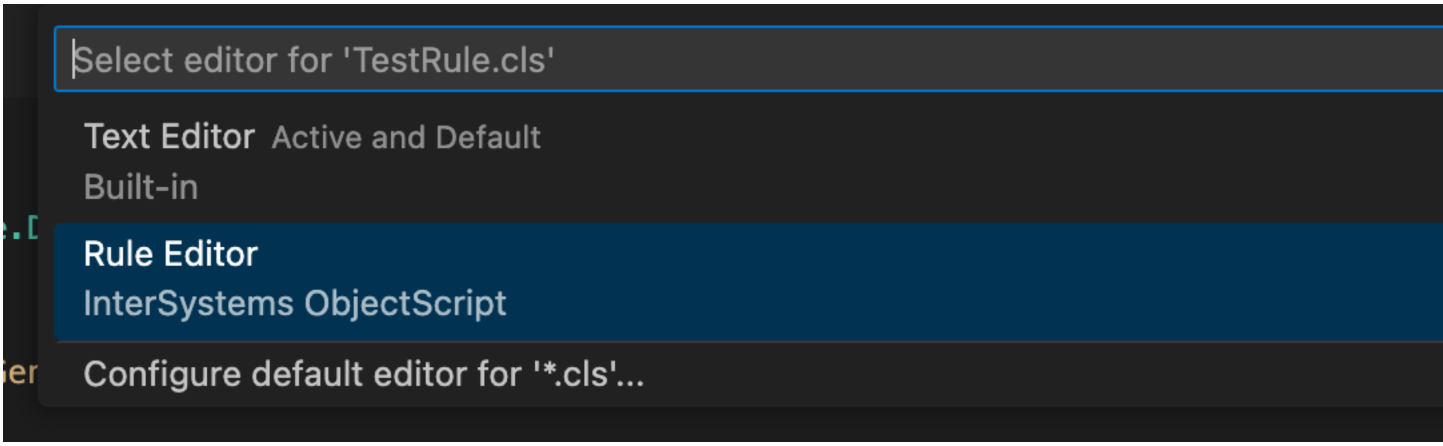
- Rule Editor (2023.1)

11.2 Opening a Low-Code Editor

To open a low-code editor, first open the class that contains the Interoperability component that you want to edit, right-click on the editor tab and select the **Reopen Editor With...** option:



You will then be prompted with a list of editors to choose from:


Select editor for 'TestRule.cls'
Text Editor Active and Default
Built-in
Rule Editor
InterSystems ObjectScript
Configure default editor for '*.cls'...

Once you select the editor, it will replace the text editor for the selected class. If the editor cannot be loaded, a modal dialog will be shown that contains the reason and the class will be automatically reopened in the default text editor. A low-code editor tab will behave the same as a text editor tab.

11.3 How They Work

This section describes how the low-code editors are integrated in VS Code to create a hassle-free editing experience. Note that while low-code editors are supported for both client-side and server-side workflows, an active server connection is required even when working client-side.

- VS Code sends your credentials to the editor so you don’t have to log in again.
- A save, undo, redo or revert action triggered by VS Code (via keyboard shortcuts, for example) will trigger the corresponding action in the editor.
- When the state of the class changes from clean to dirty (or vice versa) in the editor, the underlying text document will also be made dirty/clean.
- When the class is saved or compiled by the editor, VS Code will pull the changes from the server and update the text document.
- If the `objectscript.compileOnSave` setting is enabled and the class was saved by the editor, the class will also be compiled by the editor.

Note that the changes you make in the low-code editor are only synced to the underlying text document when you save them in the editor. Therefore, it is *strongly* recommended that you only open and edit the document in one editor (text or low-code) at once to avoid overwriting changes. The low-code editors provide support for server-side source control natively. The underlying text document is kept in sync after saves so changes can be stored in client-side source control.

12

Report an Issue

InterSystems ObjectScript for VS Code consists of three collaborating VS Code extensions. This modular architecture also means there are three different GitHub repositories where issues can be created. Fortunately, VS Code provides a convenient interface to help with the task. (Note: you must have a GitHub account to report an issue through GitHub.)

To create a GitHub issue from within the VS Code interface, perform the following steps:

1. From the **Help** menu in VS Code select **Report Issue**. Alternatively, open the Command Palette and run the **Help: Report Issue...** command.
2. When the dialog appears, use the first drop-down menu to select the category which most closely describes the issue you want to report. The menu provides the following options:
    - **Bug Report**
    - **Feature Request**
    - **Performance Issue**
3. In the second drop-down menu, select **An extension**.
4. The third drop-down menu allows you to specify one of the extensions you have installed. You can type a few characters to find the right entry. For example, `isls` quickly identifies **InterSystems Language Server**.

    Which extension should you choose? Here is a guide:

    - Select **InterSystems Language Server** for issues related to:
        - code coloring
        - Intellisense
    - Select **InterSystems ObjectScript** for issues related to:
        - export, import and compile
        - ObjectScript Explorer (browsing namespace contents)
        - direct server-side editing using isfs:// folders in a workspace
        - integration with server-side source control etc
    - Select **InterSystems Server Manager** for issues related to:
        - InterSystems Server view
        - password management in local keychain
        - definition and selection of entries in intersystems.servers

If unsure, select **InterSystems ObjectScript**.

5. Enter a descriptive one-line summary of your issue. Based on the content of your summary, the dialog suggests a list of existing issues which may be duplicates of your issue. If you don’t find an existing issue that covers yours, proceed.
6. Enter details regarding your issue. If VS Code is authenticated to GitHub, the dialog’s button is labelled **Create on GitHub**; selecting this button will open the issue on Github and then load it in your browser so that you can edit it. If VS Code is not authenticated to GitHub, the dialog’s button reads **Preview on GitHub**; selecting it launches a browser page where you must complete and submit your report.

   Here are some tips for using the GitHub page:

   - Paste images from your clipboard directly into the report field. For hard-to-describe issues, an animated GIF or a short MP4 provides invaluable help. The **Developer: Toggle Screencast Mode** feature within VS Code can help you create a recording.
   - Link to related issues by prefixing the target number with the # character.
   - Remember that whatever you post here is visible to anyone on the Internet. Mask or remove any confidential information. Be polite.

A

Settings Reference

The extensions in the InterSystems ObjectScript Extension Pack provide many settings that allow you to configure their behavior. Below you will find a table containing all settings for each extension in the pack, as well as a short description, the type of value they accept, the default value and any other notes that may be useful to you. Please see this VS Code documentation page for more information about settings and how to change them.

A.1 InterSystems Language Server

Setting	Description	Type	Default	Notes
"intersystems.language-server.diagnostics.classes"	Controls whether error diagnostics are provided when a class that is being referred to doesn't exist in the database.	`boolean`	`true`	
"intersystems.language-server.diagnostics.deprecation"	Controls whether strikethrough warning diagnostics are provided when a class or class member that is being referred to is deprecated.	`boolean`	`true`	
"intersystems.language-server.diagnostics.parameters"	Controls whether warning diagnostics are provided when a class Parameter has an invalid type or the assigned value of the Parameter doesn't match the declared type.	`boolean`	`true`	

Setting	Description	Type	Default	Notes
"intersystems.language-server.diagnostics.routines"	Controls whether error diagnostics are provided when a routine or include file that is being referred to doesn't exist in the database.	`boolean`	`false`	
"intersystems.language-server.diagnostics.suppressSyntaxErrors"	Controls the languages that syntax error diagnosics will be suppressed for.	`array`	`[]`	Each array element must be one of the following values: `"COS"`, `"SQL"`, `"CLS"`, `"HTML"`, `"PYTHON"`, `"XML"`, `"JAVA"`, `"JAVASCRIPT"`, or `"CSS"`.
"intersystems.language-server.diagnostics.zutil"	Controls whether diagnostics are provided when a deprecated or superseded **$ZUTIL** function is being called.	`boolean`	`true`	
"intersystems.language-server.formatting.commands.case"	Controls the case that ObjectScript commands will be changed to during a document formatting request.	`"upper"`, `"lower"`, or `"word"`	`"word"`	
"intersystems.language-server.formatting.commands.length"	Controls the length that ObjectScript commands will be changed to during a document formatting request.	`"short"` or `"long"`	`"long"`	
"intersystems.language-server.formatting.expandClassNames"	Controls whether short class names will be expanded to include a package during a document formatting request.	`boolean`	`false`	

Setting	Description	Type	Default	Notes
`"intersystems.language-server.formatting.system.case"`	Controls the case that ObjectScript system functions and variables will be changed to during a document formatting request.	`"upper"`, `"lower"`, or `"word"`	`"upper"`	
`"intersystems.language-server.formatting.system.length"`	Controls the length that ObjectScript system functions and variables will be changed to during a document formatting request.	`"short"` or `"long"`	`"long"`	
`"intersystems.language-server.hover.commands"`	Controls whether hover information is provided for ObjectScript commands.	`boolean`	`true`	
`"intersystems.language-server.hover.preprocessor"`	Controls whether hover information is provided for ObjectScript preprocessor directives.	`boolean`	`true`	
`"intersystems.language-server.hover.system"`	Controls whether hover information is provided for ObjectScript system functions and variables.	`boolean`	`true`	
`"intersystems.language-server.refactor.exceptionVariable"`	The name of the exception variable inserted in a ‘Wrap in Try/Catch’ refactor.	`string`	`"ex"`	
`"intersystems.language-server.signaturehelp.documentation"`	Controls whether documentation for a method is shown when a SignatureHelp is active.	`boolean`	`true`	This setting does not affect documentation for macro SignatureHelp views, which is always shown.

Setting	Description	Type	Default	Notes
`"intersystems.language-server.suggestTheme"`	Controls whether the extension will suggest that one of the InterSystems default themes be used if neither one is active upon extension activation.	`boolean`	`true`	
`"intersystems.language-server.trace.server"`	Traces the communication between VS Code and the language server.	`"off"`, `"messages"`, or `"verbose"`	`"off"`	Any trace information will be logged to the InterSystems Language Server Output channel.

A.2 InterSystems ObjectScript

Setting	Description	Type	Default	Notes
`"objectscript.autoPreviewXML"`	Automatically preview XML export files in UDL format.	`boolean`	`false`	
`"objectscript.autoShowTerminal"`	Automatically show terminal when connected to docker-compose.	`boolean`	`false`	
`"objectscript.compileFlags"`	Compilation flags.	`string`	`"cuk"`	Common compilation flags are `b` (compile dependent classes), `k` (keep generated source code) and `u` (skip related up-to-date documents). For descriptions of all available flags and qualifiers, click here.
`"objectscript.compileOnSave"`	Automatically compile an InterSystems file when saved in the editor.	`boolean`	`true`	

Setting	Description	Type	Default	Notes
`"objectscript.conn"`	Configures the active server connection.	`object`	`undefined`	See Configuring a Server Connection for more details on configuring server connections.
`"objectscript.conn.ns"`	InterSystems server's namespace to use.	`string`	`undefined`	
`"objectscript.conn.active"`	Should the connection be active on startup.	`boolean`	`false`	
`"objectscript.conn.username"`	InterSystems server's user name.	`string`	`undefined`	
`"objectscript.conn.password"`	InterSystems server's password.	`string`	`undefined`	For security reasons, InterSystems recommends that you do not specify your password in a config file.
`"objectscript.conn.server"`	InterSystems server's name in Server Manager settings from which to get connection info.	`string`	`undefined`	Specify only `"ns"` and `"active"` when using this setting. See the Server Manager README for more details.
`"objectscript.conn.docker-compose"`	Configures the active server port using information from a file which must be named docker-compose.yml in the project's root directory.	`object`	`undefined`	
`"objectscript.conn.docker-compose.service"`	InterSystems service's name in docker-compose.yml.	`string`	`undefined`	
`"objectscript.conn.docker-compose.internalPort"`	InterSystems service's internal port in docker-compose.yml.	`object`	`undefined`	

Setting	Description	Type	Default	Notes
`"objectscript.debug.copyToClipboard"`	Show inline **Copy Invocation** CodeLens action for ClassMethods and Routine Labels.	`boolean`	`true`	
`"objectscript.debug.debugThisMethod"`	Show inline **Debug** CodeLens action for ClassMethods and Routine Labels.	`boolean`	`true`	
`"objectscript.explorer.alwaysShowServerCopy"`	Always show the server copy of a document opened from the InterSystems Explorer.	`boolean`	`false`	
`"objectscript.export"`	Configures the files that the **Export Code from Server** command will export from the server to the local workspace folder.	`object`	`undefined`	
`"objectscript.export.addCategory"`	Add a category folder to the beginning of the export path.	`boolean` or `object`	`false`	
`"objectscript.export.atelier"`	Export source code as Atelier did it, with packages as subfolders.	`boolean`	`true`	This setting only affects classes, routines, include files and DFI files.
`"objectscript.export.category"`	Category of source code to export: `CLS` = classes; `RTN` = routines; `CSP` = csp files; `OTH` = other. Default is `*` = all.	`string` or `object`	`"*"`	
`"objectscript.export.dontExportIfNoChanges"`	Do not rewrite the local file if the content is identical to what came from the server.	`boolean`	`false`	

Setting	Description	Type	Default	Notes
`"objectscript.export.exactFilter"`	SQL filter to limit what to export.	`string`	`""`	The filter is applied to document names using the LIKE predicate (i.e. `Name LIKE 'exactFilter'`). If provided, `objectscript.export.filter` is ignored.
`"objectscript.export.filter"`	SQL filter to limit what to export.	`string`	`""`	The filter is applied to document names using the LIKE predicate (i.e. `Name LIKE '%filter%'`).
`"objectscript.export.folder"`	Folder for exported source code within workspace.	`string`	`"src"`	This setting is relative to the workspace folder root.
`"objectscript.export.generated"`	Export generated source code files, such as INTs generated from classes.	`boolean`	`false`	
`"objectscript.export.map"`	Map file names before export, with regexp pattern as a key and replacement as a value.	`object`	`{}`	For example, `{ \"%(.*)\": \"_$1\" }` to make % classes or routines use underscore prefix instead.
`"objectscript.export.mapped"`	Export source code files mapped from a non-default database.	`boolean`	`true`	
`"objectscript.export.maxConcurrentConnections"`	Maximum number of concurrent export connections.	`number`	`0`	`0` = unlimited
`"objectscript.export.noStorage"`	Strip the storage definition on export.	`boolean`	`false`	Can be useful when working across multiple systems.

Setting	Description	Type	Default	Notes
`"objectscript.format.commandCase"`	Case for commands.	`"upper"`, `"lower"`, or `"word"`	`"word"`	Has no effect if the InterSystems Language Server extension is installed and enabled.
`"objectscript.format.functionCase"`	Case for system functions and system variables.	`"upper"`, `"lower"`, or `"word"`	`"word"`	Has no effect if the InterSystems Language Server extension is installed and enabled.
`"objectscript.importOnSave"`	Automatically save a client-side InterSystems file on the server when saved in the editor.	`boolean`	`true`	
`"objectscript.multilineMethodArgs"`	List method arguments on multiple lines, if the server supports it.	`boolean`	`false`	Only supported on IRIS 2019.1.2, 2020.1.1+, 2021.1.0+ and subsequent versions! On all other versions, this setting will have no effect.
`"objectscript.openClassContracted"`	Automatically collapse all class member folding ranges when a class is opened for the first time.	`boolean`	`false`	
`"objectscript.overwriteServerChanges"`	Overwrite a changed server version without confirmation when importing the local file.	`boolean`	`false`	

Setting	Description	Type	Default	Notes
`"objectscript.projects.webAppFileExtensions"`	When browsing a virtual workspace folder that has a project query parameter, all files with these extensions will be automatically treated as web application files. Extensions added here will be appended to the default list and should *NOT* include a dot.	`string array`	`[]`	Default extensions: [illegible]
`"objectscript.serverSourceControl.disableOtherActionTriggers"`	Prevent server-side source control 'other action' triggers from firing.	`boolean`	`false`	
`"objectscript.showExplorer"`	Show the InterSystems Explorer view.	`boolean`	`true`	
`"objectscript.showGeneratedFileDecorations"`	Controls whether a badge is shown in the file explorer and open editors view for generated files.	`boolean`	`true`	
`"objectscript.showProposedApiPrompt"`	Controls whether a prompt to enable VS Code proposed APIs is shown when a server-side workspace folder is opened.	`boolean`	`true`	
`"objectscript.studioActionDebugOutput"`	Log the action that VS Code should perform as requested by the server, in JSON format.	`boolean`	`false`	Actions will be logged to the ObjectScript Output channel.
`"objectscript.suppressCompileErrorMessages"`	Suppress popup messages about errors during compile, but still focus on Output view.	`boolean`	`false`	

Setting	Description	Type	Default	Notes
"objectscript.suppressCompileMessages"	Suppress popup messages about successful compile.	`boolean`	`true`	
"objectscript.unitTest.autoload.folder"	Customize the name of the subdirectory used to autoload resources which are required by client-side unit tests.	`string`	`"_autoload"`	
"objectscript.unitTest.autoload.udl"	Control whether the autoload feature for client-side unit tests loads UDL files (.cls, .mac, .int, .inc).	`boolean`	`true`	
"objectscript.unitTest.autoload.xml"	Control whether the autoload feature for client-side unit tests loads XML files.	`boolean`	`true`	
"objectscript.unitTest.relativeTestRoots"	Specify paths to where client-side test classes are stored, relative to the workspace's root directory.	`string array`	`[]`	
"objectscript.unitTest.showOutput"	Control whether console output is shown for unit tests.	`boolean`	`true`	
"objectscript.webSocketTerminal.syntaxColoring"	Enable syntax coloring for command input in the InterSystems WebSocket Terminal.	`boolean`	`true`	

A.3 InterSystems Server Manager

Setting	Description	Type	Default	Notes
`"intersystems.servers"`	InterSystems servers that other extensions connect to. Each property of this object names a server and holds nested properties specifying how to connect to it.	`object`	`undefined`	See Configuring a Server for more details on configuring servers.