AppMill

PLUGINS AND ALIAS FRAMEWORK EXTENSION FUNCTIONS DEBUGGING GUIDE

Table of Contents

Revisions

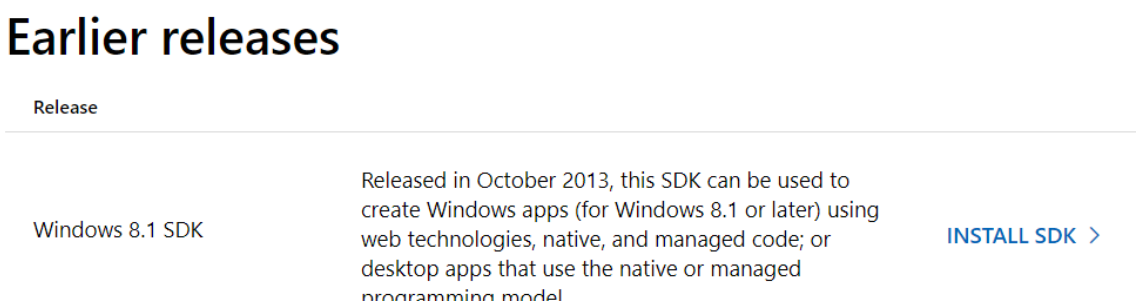
Revision	Date	Description
0.1	06/02/21	Initial revision: • Added debugging instructions for C++, Python, JavaScript, PHP, and C#.
0.2	06/10/21	• Added php.ini settings.
0.3	04/02/24	• Added remote debugging instructions for C++ Python, JavaScript, PHP, and C#.

1 Debugging C++ plugin

1.1 Required tools

Download and install the tools below:

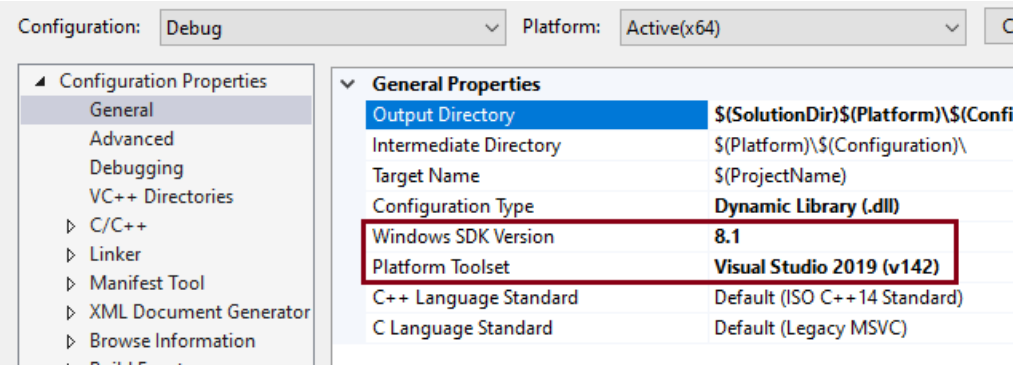
- **Visual Studio 2019**
  The Visual Studio 2019 tool can be downloaded at the following link:
  https://visualstudio.microsoft.com/downloads
  - Install **Desktop development with C++**.
- **Windows 8.1 SDK for MS Windows**
  The Windows 8.1 SDK for MS Windows tool can be downloaded at the following link:
  https://developer.microsoft.com/en-us/windows/downloads/sdk-archive
  - In the **Earlier releases** section, download the **Windows 8.1 SDK** release by clicking **the Install SDK** link.



1.2 Tool configurations

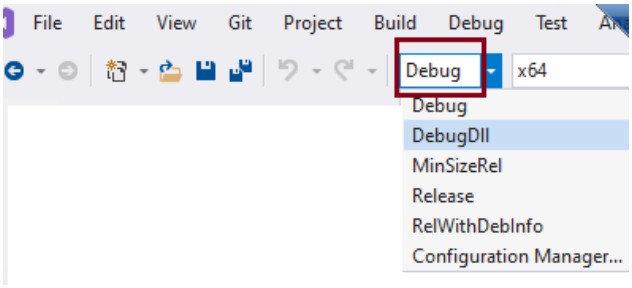
1.2.1 Project configuration

In Visual Studio Solution Explorer, right click on the project and select Properties to view the project properties. The C++ plugin project must be built with the Windows 8.1 SDK and v142 platform toolset.



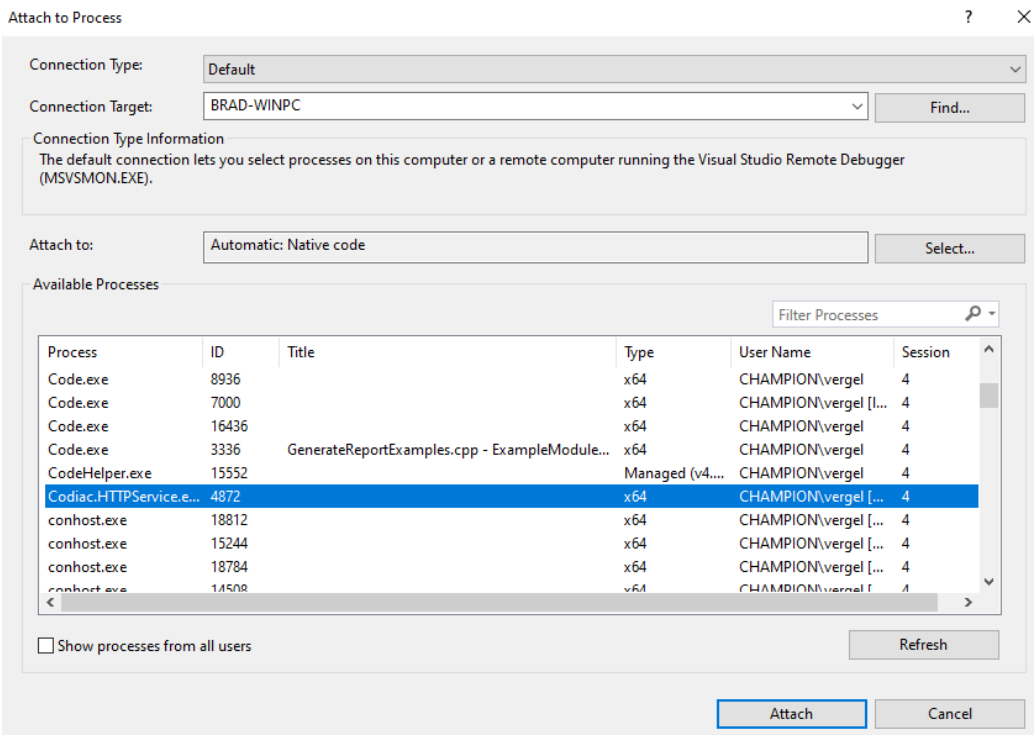
1.2.2 Build C++ plugin with debug symbols

Select the Debug configuration and build C++ plugin project (Ctrl-Shift-B). Then, copy the output DLL files to *<service root>\Debug\Modules* directory.

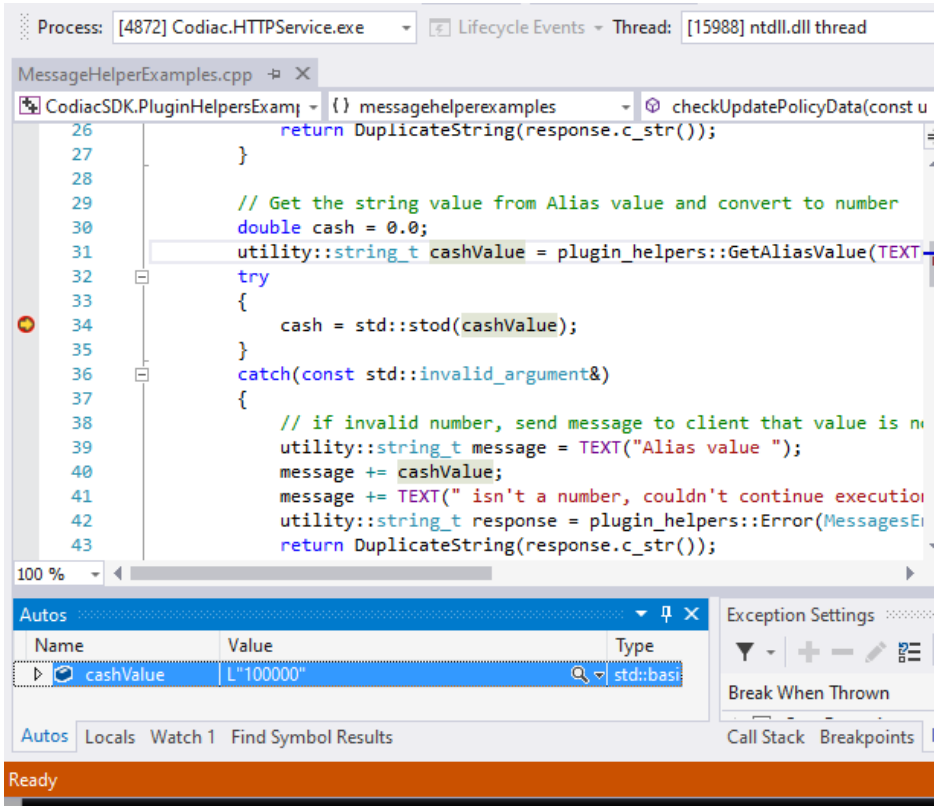


1.3 Debugging

1. Run the service Debug build in administrator account.
2. Run the Visual Studio in administrator account.
   - Visual Studio must be run with the same account as service.
3. Open the C++ plugin project.
4. Attach Visual Studio debugger to **Codiac.HTTPService.exe** process (Ctrl-Alt-P).



5. Open the plugin class and set the breakpoint.
6. Execute the extension function in the client. This should stop at the breakpoint, and the current line will have a yellow arrow.



7. You can step through the code by pressing the F10 (step over) and F11 (step into) keys and see the variable values in watches.
8. To end debugging, open the **Debug** menu and select the **Detach all** menu item.

2 Debugging C++ AF extension module

2.1 Required tools

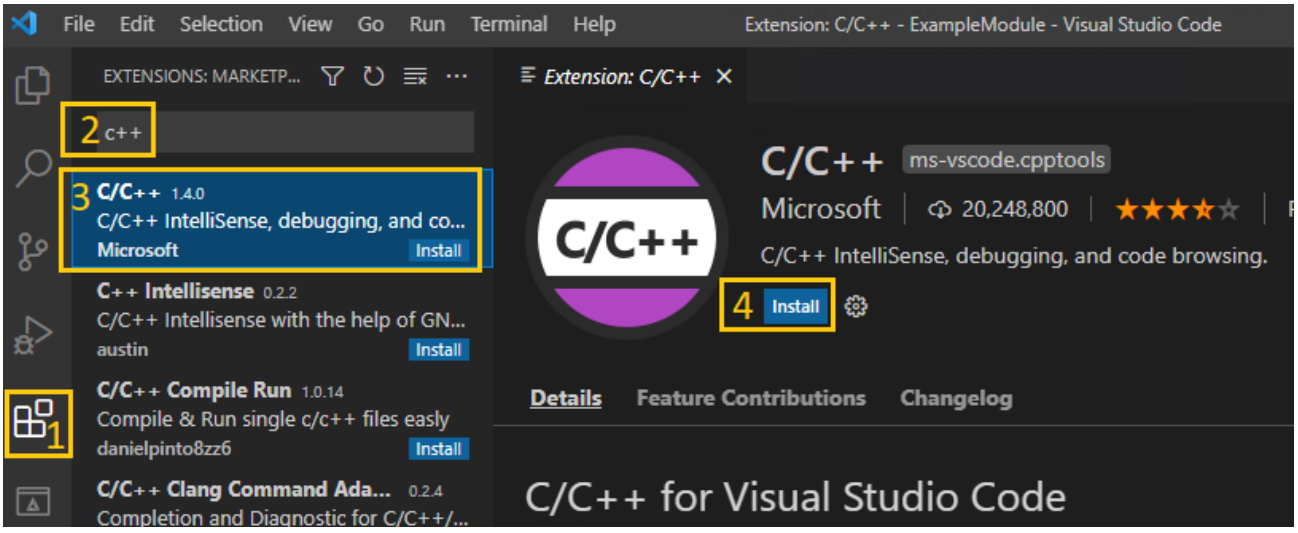
Download and install the tools below:

- **Visual Studio Code**
  The Visual Studio Code tool can be downloaded at the following link:
  https://code.visualstudio.com/Download
- **Cmake**
  The Cmake tool can be downloaded at the following link: https://cmake.org/download/
  - for building C++ AF extension module.

2.2 Tool configurations

2.2.1 Install C++ extension for Visual Studio Code

1. In Visual Studio Code, click the extension icon on left side bar.
2. Enter **C++** to search the extensions.
3. Select **C/C++** extension by Microsoft.
4. Click the **Install** button to start the installation.

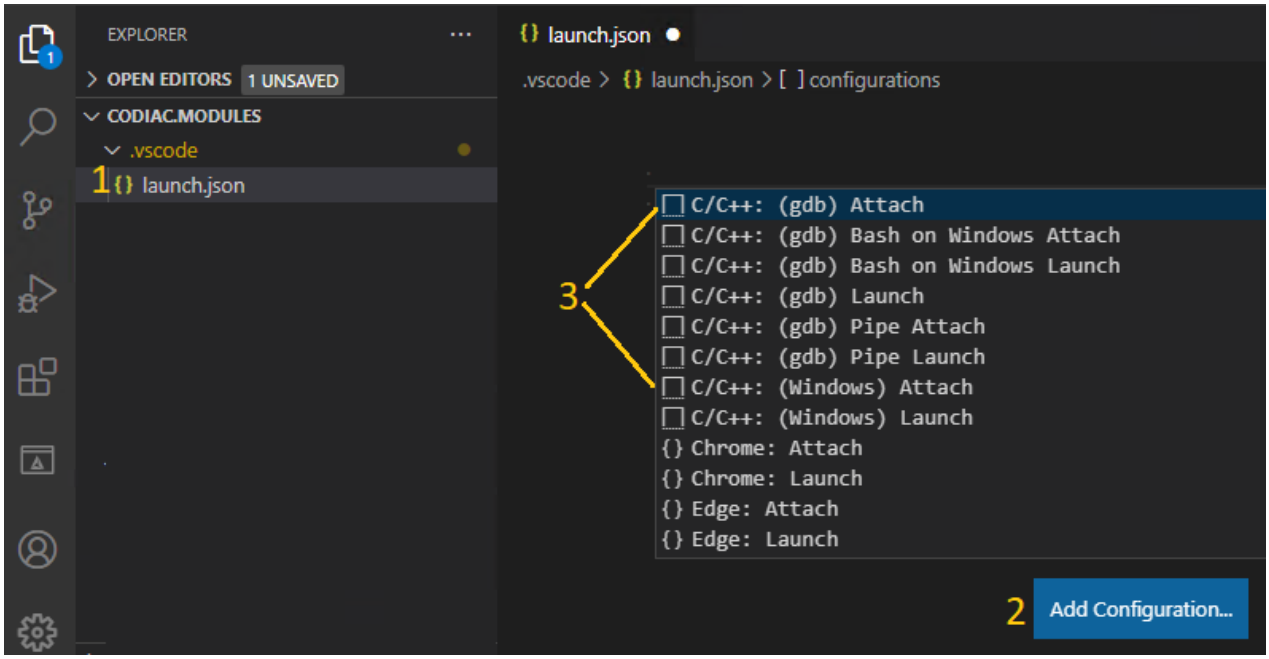


2.2.2 Create launch.json for VS Code debugger

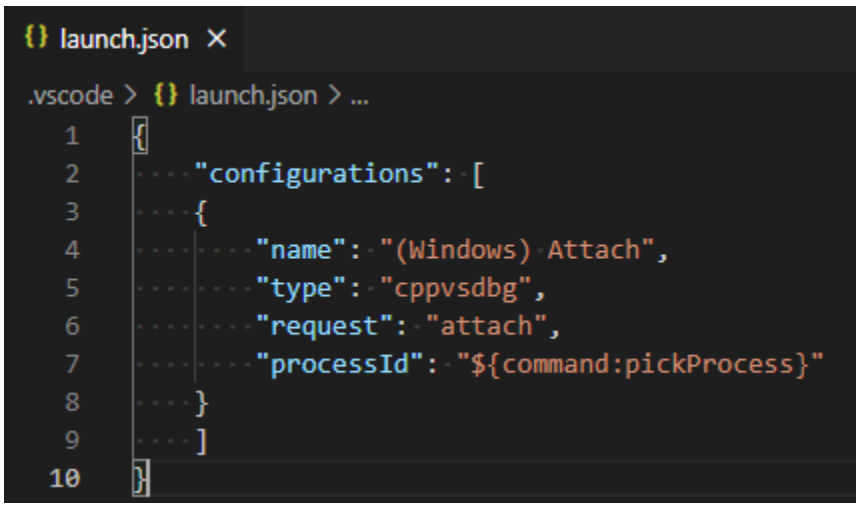
The **launch.json** file contains configuration for how VS Code starts the debugger. For debugging the AF extension module, the VS code debugger needs to be attached to the Codiac service process.

The Codiac module project template contains **.vscode\launch.json** file that enables VS Code to attach to the application process. This can be recreated by following the instructions below:

1. Create **launch.json** file (if not created yet) in the **.vscode** directory.
2. Click the **Add Configuration...** button.
3. Select **C/C++: (Windows) Attach** if running on MS Windows or **C|C++: (gdb) Attach** if running on Linux.



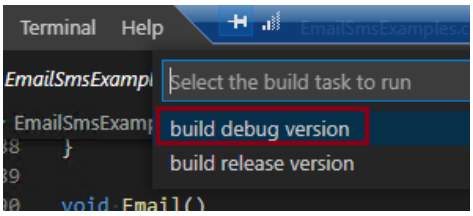
4. This will create **(Windows) Attach** configuration shown below.



5. Save the **launch.json** file.

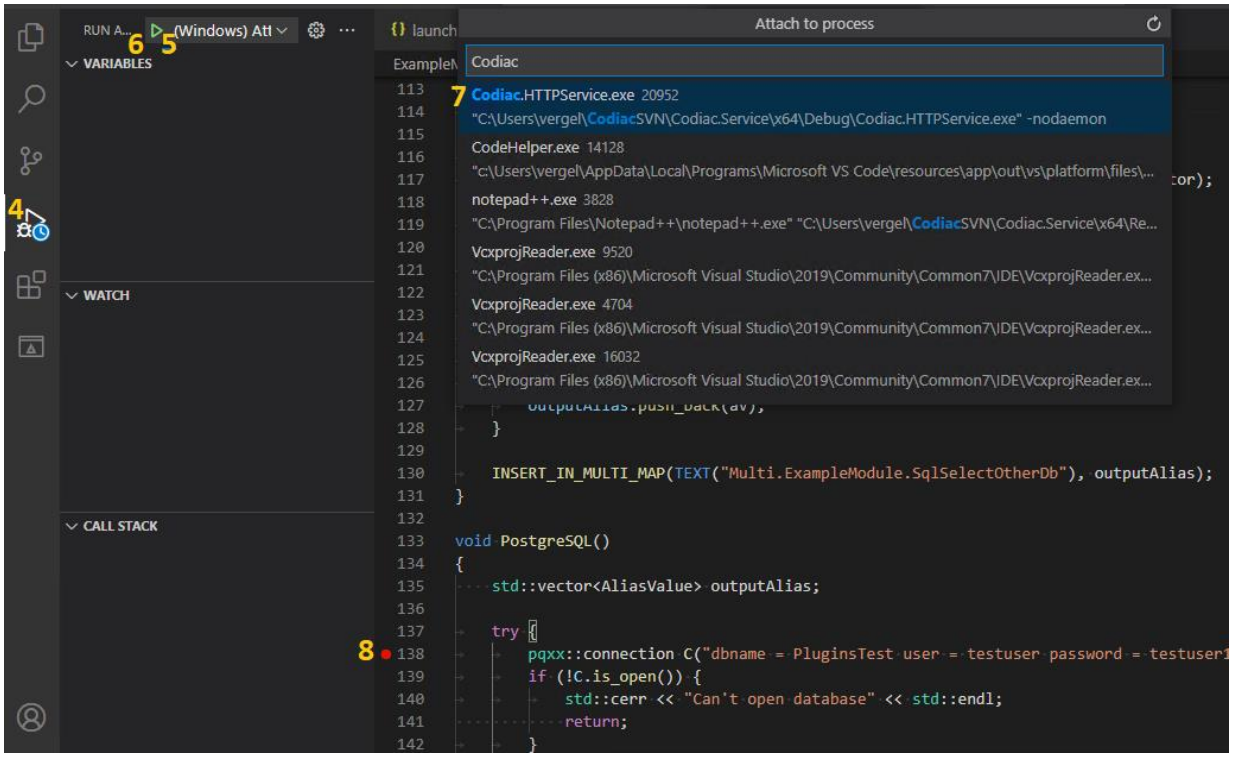
2.2.3 Build C++ AF extension module with debug symbols

In VS Code, press the **Ctrl-Shift-B** key combination and select **build debug version**. Then, copy the debug build output files (which is *build\Modules\Debug directory*) to *<service root>\Debug\Modules* directory.

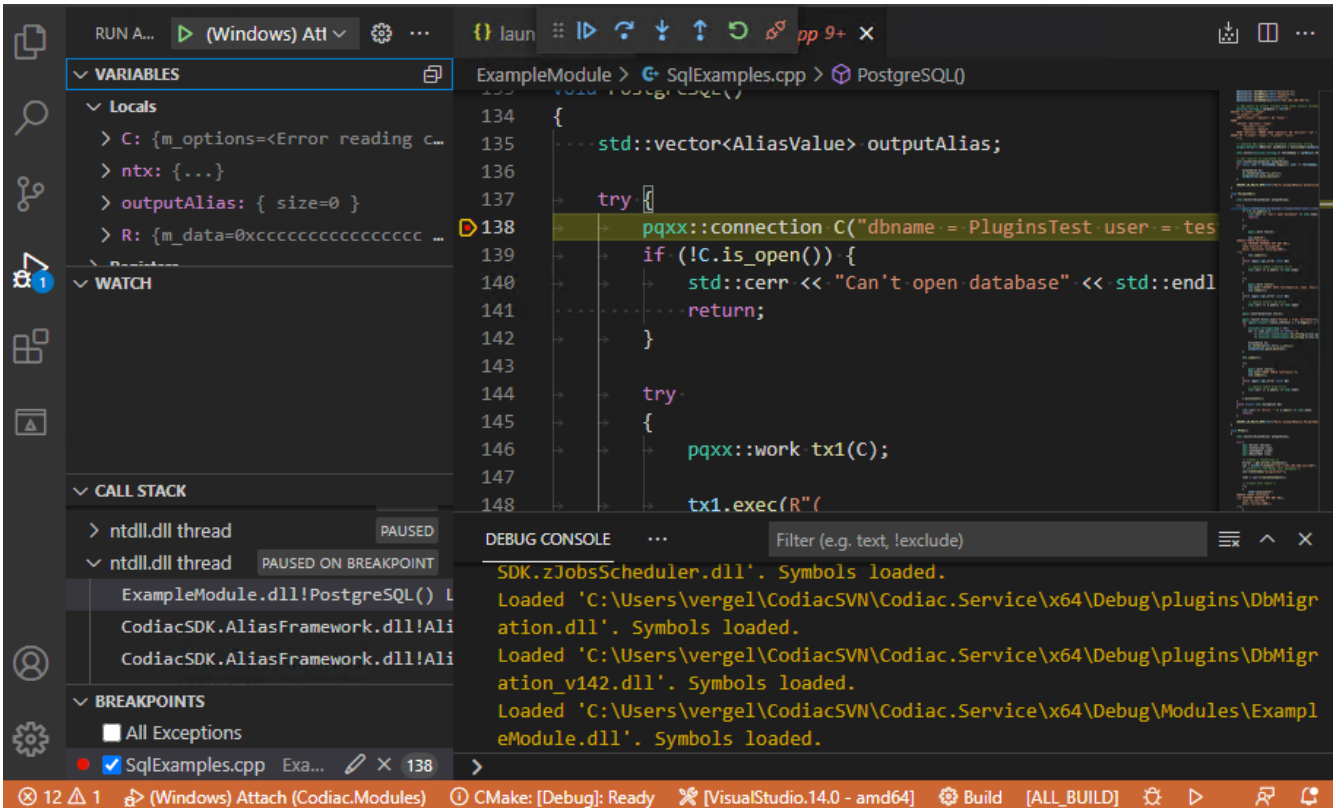


2.3 Debugging

1. Run the service Debug build in administrator account.
   - Do not use service Release build for debugging AF extension module.
2. Run the VS Code in administrator account.
   - The VS Code must be run with the same account as service.
3. Open the source code directory of AF extension module.
4. Click the debug icon on the left side bar.
5. Select **Windows (Attach)**.
6. Click the green play button (or F5) to start debugging.
7. Select the **Codiac.Service** process.
   - At this point, VS Code is attached to the service process, and the bottom part turns orange.
8. Set a breakpoint on the AF extension function.



9. Execute a client request that evaluates the custom alias.
    - This custom alias executes the AF extension function.
10. VS Code should stop at the breakpoint and highlight the current line.
11. At this point, you can step through the codes by pressing the F10 (step next), F11 (step into), and F5 (continue) keys or disconnect (orange disconnect icon). You can add watches or see local watches.


RUN A...
(Windows) Atl
VARIABLES
Locals
C: {m_options=<Error reading c...
ntx: {...}
outputAlias: { size=0 }
R: {m_data=0xcccccccccccccccc ...
WATCH
CALL STACK
ntdll.dll thread PAUSED
ntdll.dll thread PAUSED ON BREAKPOINT
ExampleModule.dll!PostgreSQL() L
CodiacSDK.AliasFramework.dll!Ali
CodiacSDK.AliasFramework.dll!Ali
BREAKPOINTS
All Exceptions
SqlExamples.cpp Exa... 138
ExampleModule > SqlExamples.cpp > PostgreSQL()
134 {
135 std::vector<AliasValue> outputAlias;
136
137 try {
138 pqxx::connection C("dbname = PluginsTest user = tes
139 if (!C.is_open()) {
140 std::cerr << "Can't open database" << std::endl
141 return;
142 }
143
144 try
145 {
146 pqxx::work tx1(C);
147
148 tx1.exec(R"(
DEBUG CONSOLE
Filter (e.g. text, !exclude)
SDK.zJobsScheduler.dll'. Symbols loaded.
Loaded 'C:\Users\vergel\CodiacSVN\Codiac.Service\x64\Debug\plugins\DbMigration.dll'. Symbols loaded.
Loaded 'C:\Users\vergel\CodiacSVN\Codiac.Service\x64\Debug\plugins\DbMigration_v142.dll'. Symbols loaded.
Loaded 'C:\Users\vergel\CodiacSVN\Codiac.Service\x64\Debug\Modules\ExampleModule.dll'. Symbols loaded.
12 1
(Windows) Attach (Codiac.Modules)
CMake: [Debug]: Ready
[VisualStudio.14.0 - amd64]
Build
[ALL_BUILD]

3 Remote Debugging C++ plugin and AF extension functions

Full instruction for the remote debugging for the C++ applications can be found on the Remote Debug a C++ Project - Visual Studio (Windows) | Microsoft Learn page.

3.1 Required tools

Download and install the tools below on the remote machine:

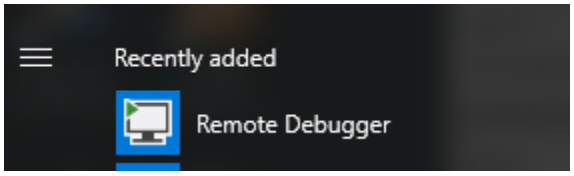
- **Remote Debugger**
  The Remote Debugger can be downloaded at the following link: Remote Debug a C++ Project - Visual Studio (Windows) | Microsoft Learn

3.2 Tool configurations

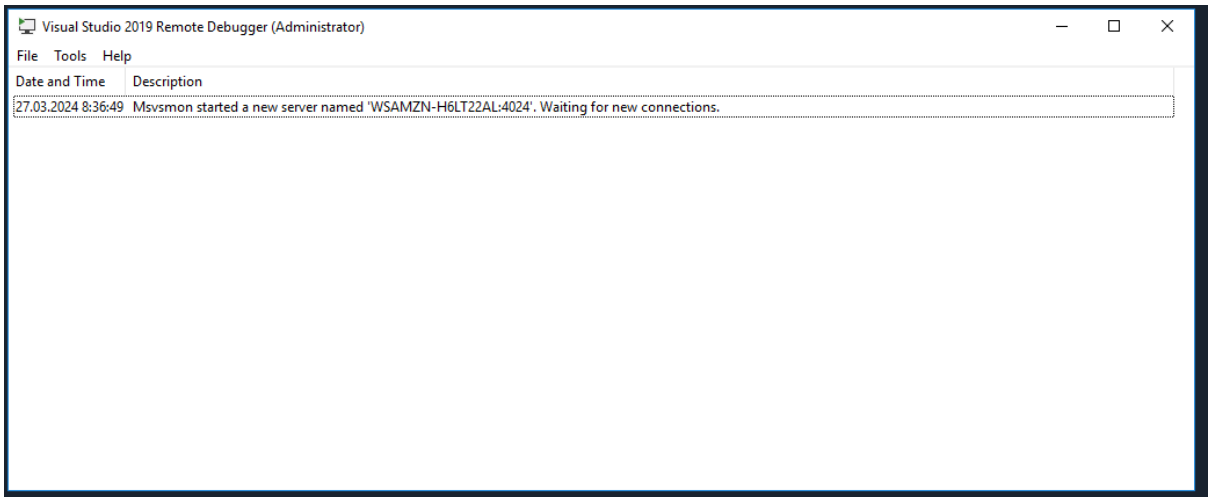
3.2.1 Install Remote Debugger

To set up the tool for remote debugging, follow the steps below:

1. Download the Remote Debugger.
2. Install the Remote Debugger on the remote machine.
3. Launch the Remote Debugger on the remote machine.

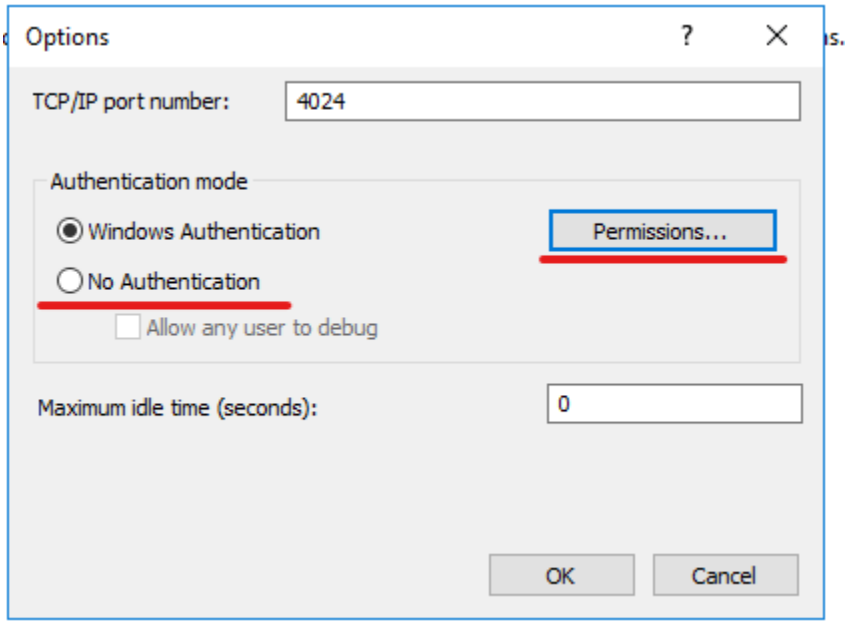


The Remote Debugger working screen will be displayed as follows:



In the Remote Debugger working screen, perform the following actions, to configure the access to the Remote Debugger:

1. In the Menu, click the **Tools** menu section and select the **Options…** menu item. The Options pop-up window opens.



In the Options pop-up window, fill in the necessary fields:

- **TCP/IP port number** – the communication endpoints that applications use to coordinate and establish communication over a network using the TCP/IP protocol suite.

Based on the desired security level select:

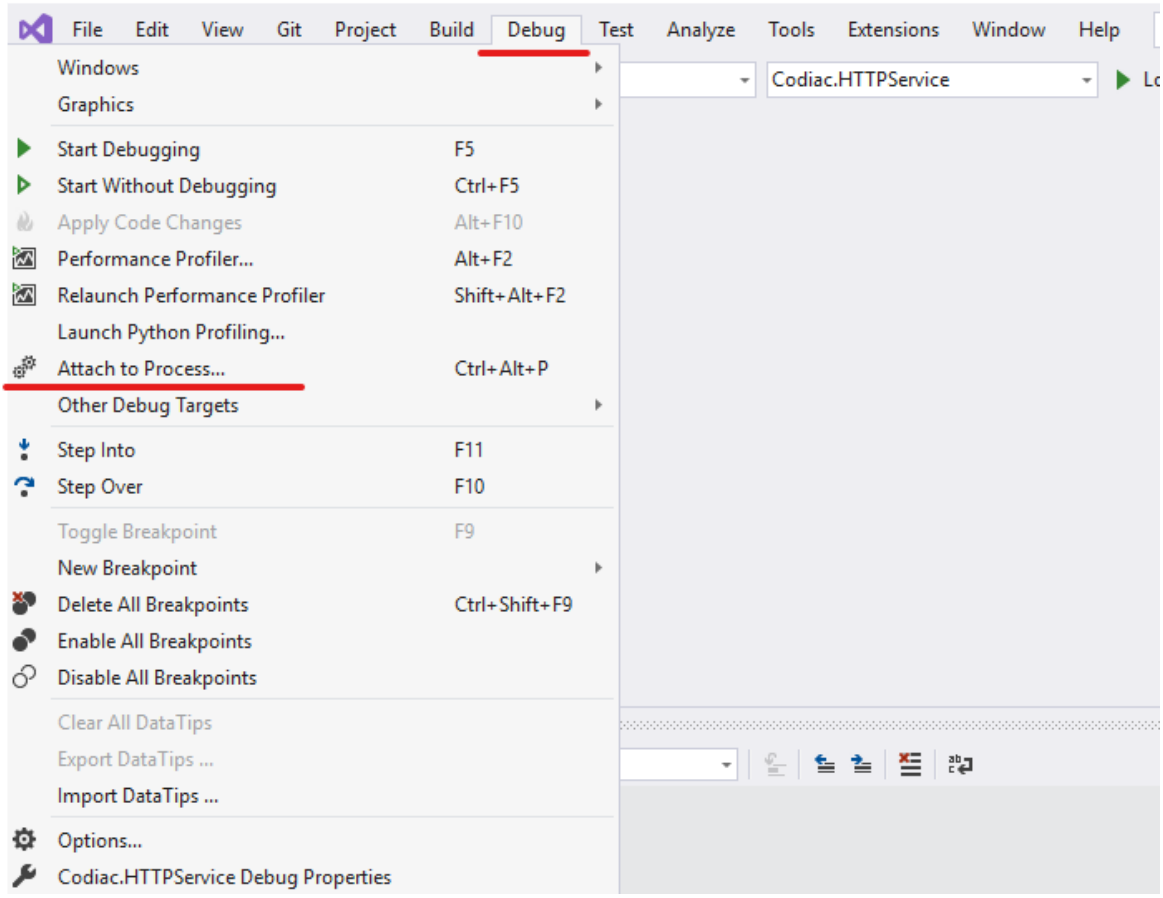
- the **Windows Authentication** access and set the **Permissions** to a user,
  or
- the **No Authentication** access for the remote debugging.

After the above-mentioned options are set, the remote machine will be ready to accept the Remote Debugger's requests.

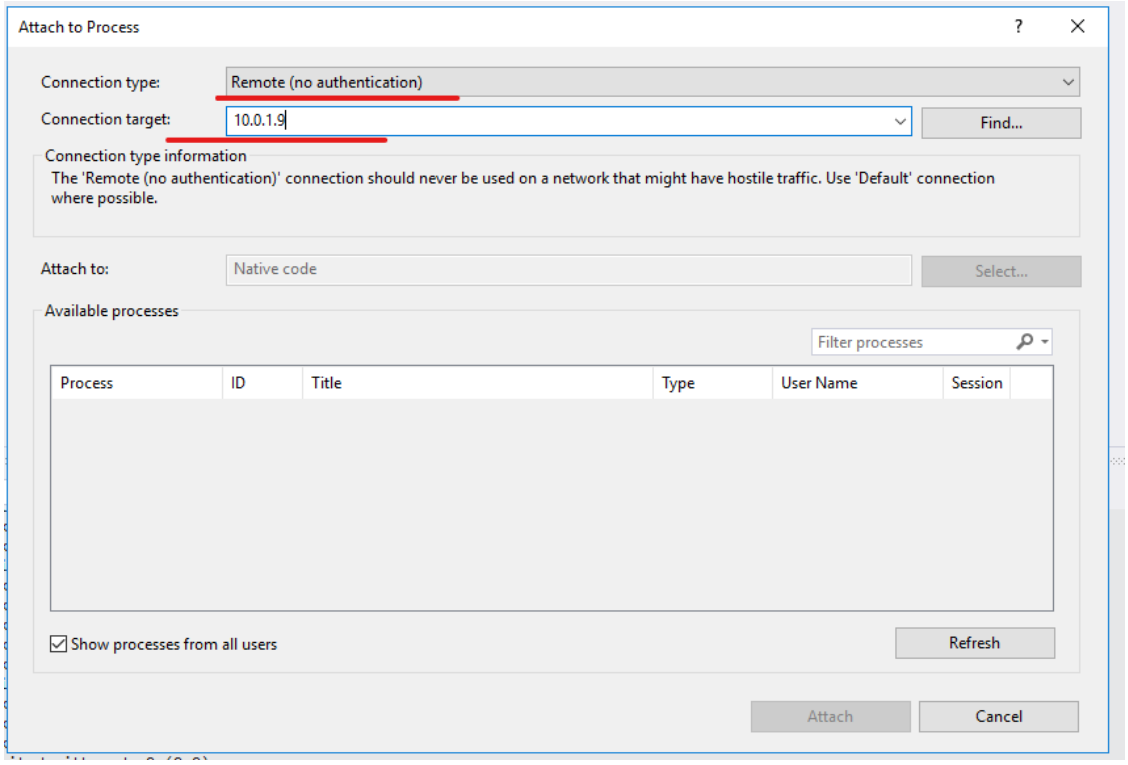
3.3 Debugging

On the local machine where plugins or modules were created, users just need to select the process in the Visual Studio. The Remote Debugger will be attached to the selected process. For that:

1. Launch Visual Studio.
2. Click the **Debug** section on the menu, and select the **Attach to Process…** menu item.



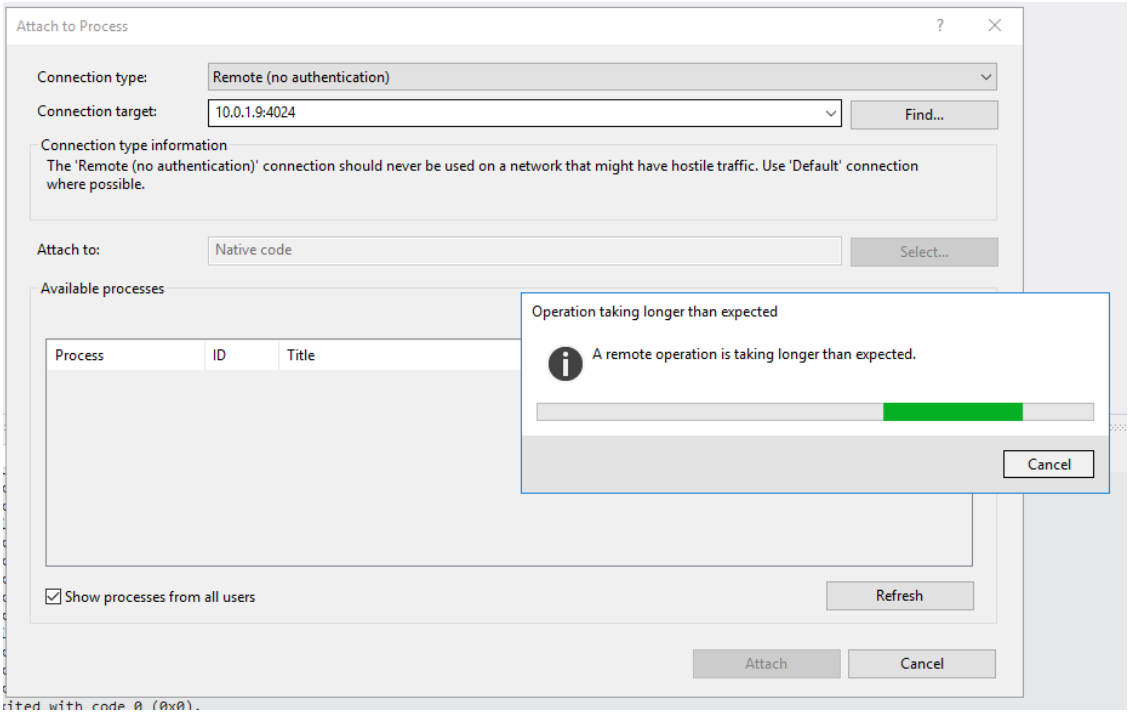
The **Attach to Process** pop-up window opens.



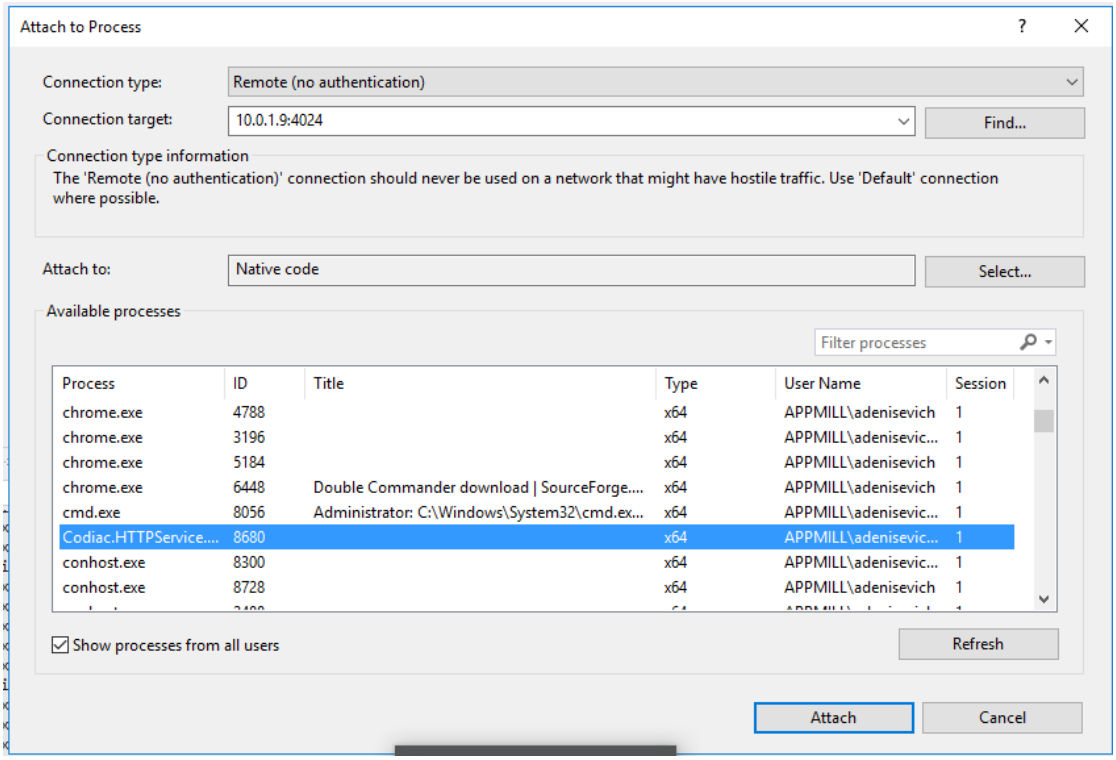
3. In the opened **Attach to Process** pop-up window, select:

- the **Remote (No authentication)** debugger type from the Connection type drop-down list,
- the remote machine IP address from the Connection target drop-down list, and
- press the ENTER key.

The debugger will connect to the remote machine, and the list of available processes that the debugger can be attached to will be shown.



4. In the available processes list, select the **Codiac.HTTPService.exe** process, and click the **Attach** button.



At reaching a code for the plugins, the process will be stopped at the breakpoint that was set in the developed plugin or modules.

To allow the remote debugging, files with the Debug information that were generated at the build should be added.

4 Debugging Python plugin and AF extension functions

4.1 Required tools

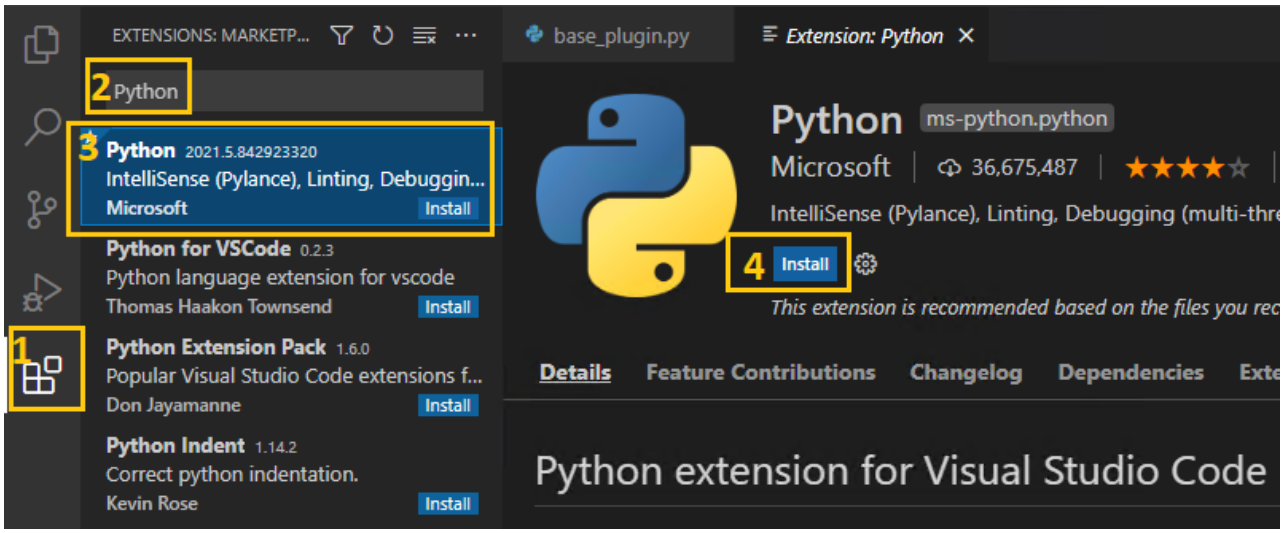
Download and install the tool below:

- **Visual Studio Code**
  The Visual Studio Code tool can be downloaded at the following link:
  https://code.visualstudio.com/Download

4.2 Tool configurations

4.2.1 Install Python extension for Visual Studio Code

1. In Visual Studio Code, click the extension icon on the left side bar.
2. Enter **Python** to search the extensions.
3. Select **Python** extension by Microsoft.
4. Click the **Install** button to start the installation.



4.2.2 Create launch.json for VS Code debugger

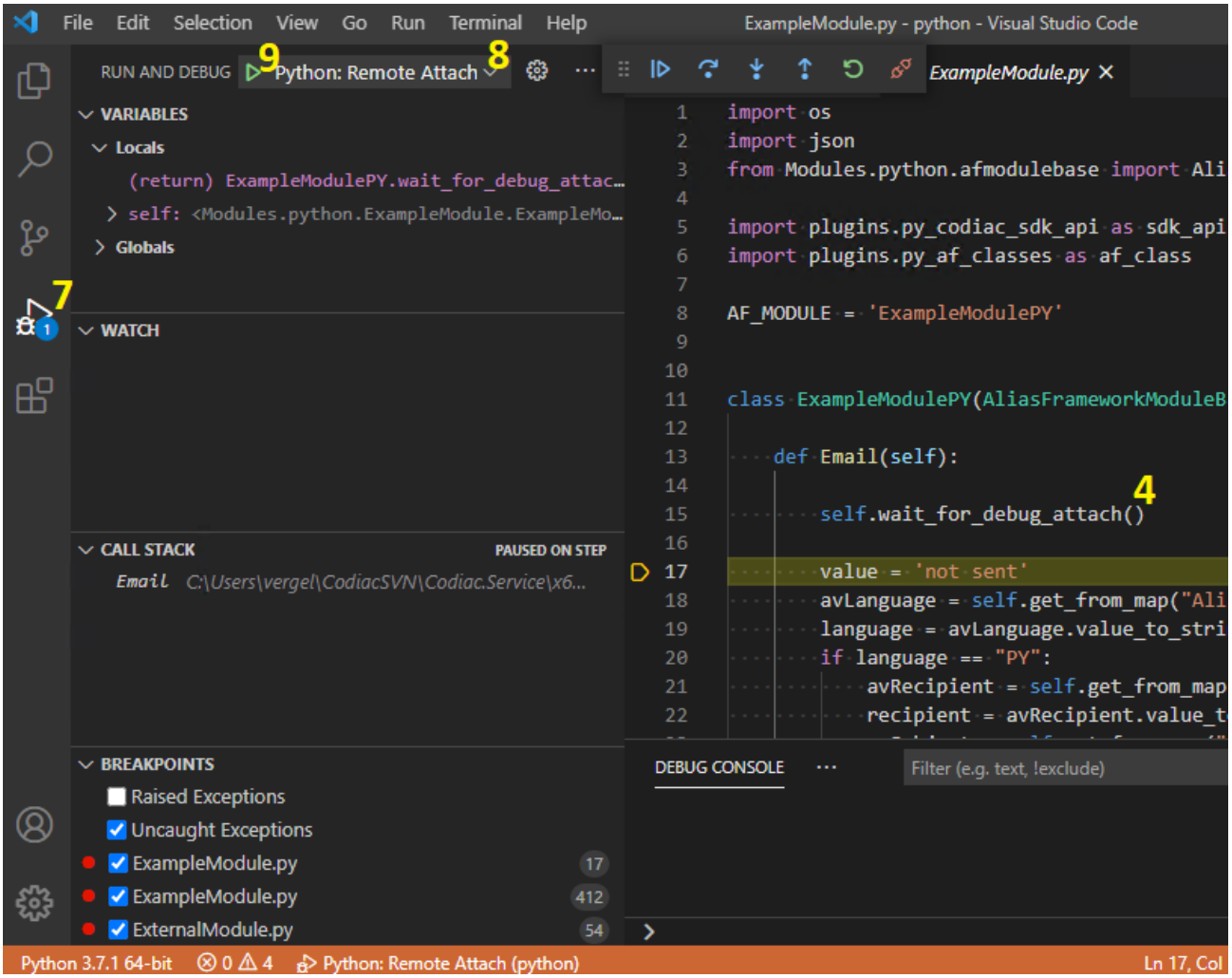
The **launch.json** file contains configuration for how VS Code starts the debugger. For debugging Python plugin or extension module, the VS code debugger needs to be attached to debugpy port 3000. Below are the instructions on how to create a **launch.json** file.

1. Create the **launch.json** file (if not created yet) in the **.vscode** directory.
2. Paste the content below into the **launch.json** file and save.

```
{
  // Use IntelliSense to learn about possible attributes.
  // Hover to view descriptions of existing attributes.
  // For more information, visit: https://go.microsoft.com/fwlink/?linkid=830387
  "version": "0.2.0",
  "configurations": [
    {
      "name": "Python: Remote Attach",
      "type": "python",
      "request": "attach",
      "connect": {
        "host": "127.0.0.1",
        "port": 3000
      },
      "pathMappings": [
        {
          "localRoot": "${workspaceFolder}",
          "remoteRoot": "."
        }
      ],
      "justMyCode": false
    }
  ]
}
```

4.3 Debugging

1. Run the service in administrator account.
2. Run the VS Code in administrator account.
   - The VS Code must be run with the same account as service.
3. Open the Python directory to be debugged.
   - AF extension function: *<service root>\Modules\python*.
   - plugins: *<service root>\plugins\python*.
4. Add a breakpoint by inserting "*self.wait_for_debug_attach()*", and save the file.
5. Execute the **Python plugin** or **Python AF extension function**.
6. Wait for console to print **waiting for debugger**.
7. Click the debug icon on the left side bar.
8. Select **Python: Remote Attach**.
9. Click the green play button (or F5) to start debugging.
10. At this point, VS Code is attached to the debugger and shows the current being executed, and the bottom part turns orange.



11. You can step through the codes by pressing the F10 (step next), F11 (step into), F5 (continue) or disconnect (orange disconnect icon) keys. You can add watches or see local watches.

5 Remote Debugging Python plugin and AF extension functions

In the Python language, remote debugging is almost identical to local debugging.
In the installed the **Python Visual Studio Debugger** engine (ptvsd), updated the following code line:

ptvsd.enable_attach(address=('localhost', 3000))

Where the **localhost** and local **port** values should be changed to the remote machine IP address and port.
Then, perform the debugging steps as for the local debugging described above in the Debugging section.

6 Debugging JavaScript plugin and AF extension functions

6.1 Required tools

Download and install the tool below:

- **Visual Studio Code**
  The Visual Studio Code tool can be downloaded at the following link:
  https://code.visualstudio.com/Download

6.2 Tool configurations

6.2.1 Create launch.json for VS Code debugger

The **launch.json** file contains configuration for how VS Code starts the debugger. For debugging JavaScript plugin and extension module, the VS code debugger needs to be attached to the inspector port **9229**. Below are the instructions on how to create launch.json file.

1. Create the **launch.json** file (if not created yet) in the **.vscode** directory.
2. Paste the content below to the **launch.json** file and save.

```
{
  // Use IntelliSense to learn about possible attributes.
  // Hover to view descriptions of existing attributes.
  // For more information, visit: https://go.microsoft.com/fwlink/?linkid=830387
  "version": "0.2.0",
  "configurations": [
    {
      "type": "node",
      "request": "attach",
      "name": "Attach",
      "port": 9229,
      "localRoot": "${workspaceFolder}",
      "remoteRoot": "."
    }
  ]
}
```

6.3 Debugging

1. Run the service in administrator account.
2. Run VS Code in administrator account.
   - VS Code must be run with the same account as service.

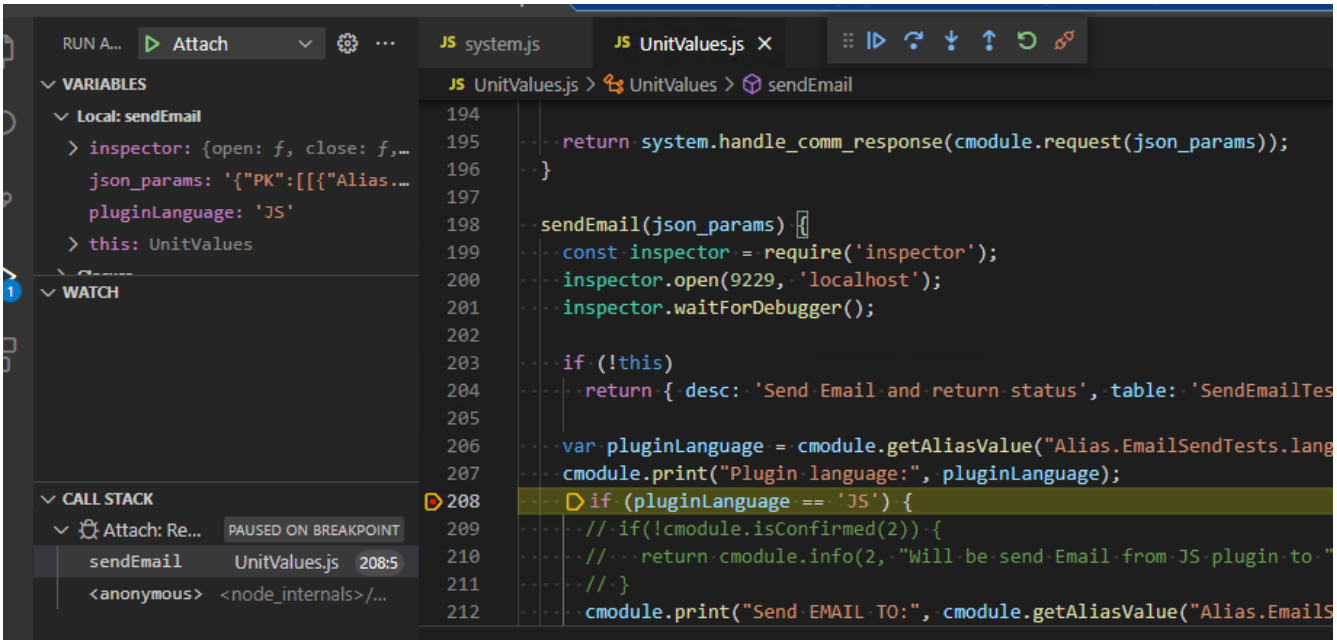
3. Open the JavaScript directory to be debugged.
   - AF extension function: *<service root>\Modules\js.*
   - plugins: *<service root>\plugins\js.*
4. Insert nodejs inspector (red box) in JavaScript script to be debugged. The port must match the port of **launch.json** file.

```
497  ExternalDb () {
498      const inspector = require('inspector');
499      inspector.open(9229, 'localhost');
500      inspector.waitForDebugger();
501
502      const fs = require('fs');
503
504      return extModule.SaveCrunchfToOtherDb('crunchf_example.db', this.getCrunchF())
505          .then(() => extModule.GetCrunchfFromOtherDb('crunchf_example.db', '10SP'))
506          .then(function(crunchfList) {
```

5. After the inspector, add breakpoints by clicking on the left side of line number.
6. Execute the **JavaScript plugin** or **JavaScript AF extension function**.
7. When the inspector starts listening, the console output will have **Debugger listening ..** as shown below:

```
Debugger listening on ws://localhost:9229/d2817c4c-9cad-4cc9-aba5-0e6adb9f5df5
Debugger listening on ws://localhost:9229/d2817c4c-9cad-4cc9-aba5-0e6adb9f5df5
For help, see: https://nodejs.org/en/docs/inspector
```

8. Click the debug icon on the left side bar.
9. Select **Attach**.
10. Click the green play button (or F5) to start debugging. At this point, VS Code is attached to the debugger.
11. You can step through the codes by pressing the F10 (step next), F11 (step into), and F5 (continue) keys or detach (orange chain icon). You can add watches or see local watches.


RUN A...
Attach
VARIABLES
Local: sendEmail
inspector: {open: ƒ, close: ƒ,...
json_params: '{"PK":[[{"Alias....
pluginLanguage: 'JS'
this: UnitValues
WATCH
CALL STACK
Attach: Re...
PAUSED ON BREAKPOINT
sendEmail
UnitValues.js
208:5
<anonymous>
<node_internals>/...
system.js
UnitValues.js
UnitValues.js > UnitValues > sendEmail
194
195 return system.handle_comm_response(cmodule.request(json_params));
196 }
197
198 sendEmail(json_params) {
199 const inspector = require('inspector');
200 inspector.open(9229, 'localhost');
201 inspector.waitForDebugger();
202
203 if (!this)
204 return { desc: 'Send Email and return status', table: 'SendEmailTes
205
206 var pluginLanguage = cmodule.getAliasValue("Alias.EmailSendTests.lang
207 cmodule.print("Plugin language:", pluginLanguage);
208 if (pluginLanguage == 'JS') {
209 // if(!cmodule.isConfirmed(2)) {
210 // return cmodule.info(2, "Will be send Email from JS plugin to "
211 // }
212 cmodule.print("Send EMAIL TO:", cmodule.getAliasValue("Alias.EmailS

7 Remote Debugging JavaScript plugin and AF extension functions

In the JavaScript language, remote debugging is almost identical to local debugging.
In the Remote Debugger in the **Attach to Process** step, select the IP address from the **Connection target** drop-down list. The IP address should belong to the remote machine.
Then, perform the debugging steps as for the local debugging described above in the Debugging section.

8 Debugging PHP plugin and AF extension functions

8.1 Required tools

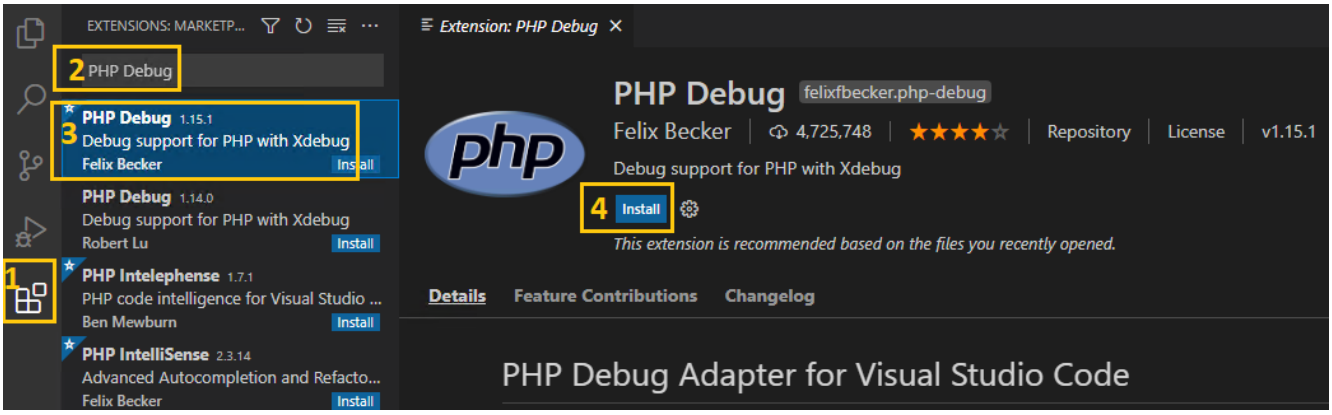
Download and install the tool below:

- **Visual Studio Code**
  The Visual Studio Code tool can be downloaded at the following link:
  https://code.visualstudio.com/Download

8.2 Tool configurations

8.2.1 Install PHP Debug extension for Visual Studio Code

1. In Visual Studio Code, click the extension icon on the left side bar.
2. Enter **PHP Debug** to search the extensions.
3. Select the **PHP Debug** extension by Felix Becker.
4. Click the **Install** button to start the installation.



8.2.2 Create launch.json for VS Code debugger

The **launch.json** file contains configuration for how VS Code starts the debugger. For debugging the PHP plugin and extension module, the VS code debugger needs to listen for an XDebug connection at port 9005. Below are the instructions on how to create a launch.json file.

1. Create the **launch.json** file (if not created yet) in the **.vscode** directory.
2. Paste the content below to the **launch.json** file and save.

```
{
  // Use IntelliSense to learn about possible attributes.
```

```
// Hover to view descriptions of existing attributes.
// For more information, visit: https://go.microsoft.com/fwlink/?linkid=830387
"version": "0.2.0",
"configurations": [
  {
    "name": "Listen for XDebug",
    "type": "php",
    "request": "launch",
    "port": 9005,
    "hostname": "localhost"
  }
]
}
```

8.2.3 Configure php.ini

XDebug extension needs to be added and configured in **php.ini** which is located in service root directory. Below are the default settings:

```
php.ini
1   html_errors=0
2   register_argc_argv=1
3   implicit_flush=1
4   output_buffering=0
5   max_execution_time=0
6   max_input_time=-1
7   extension=C:\Codiac.Service\x64\Release\CodiacSDK.AliasFramework.dll
8   extension=C:\Codiac.Service\x64\Release\plugins\CodiacSDK.PhpWrapper.dll
9   extension=C:\Codiac.Service\x64\Release\php\ext\php_sqlsrv_73_ts_x64.dll
10  extension=C:\Codiac.Service\x64\Release\php\ext\php_oci8_12c.dll
11  zend_extension=C:\Codiac.Service\x64\Release\php\ext\php_xdebug.dll
12  ;[XDebug]
13  xdebug.mode=debug
14  xdebug.start_with_request=yes
15  xdebug.client_host=127.0.0.1
16  xdebug.client_port=9005
17  xdebug.remote_handler=dbgp
```

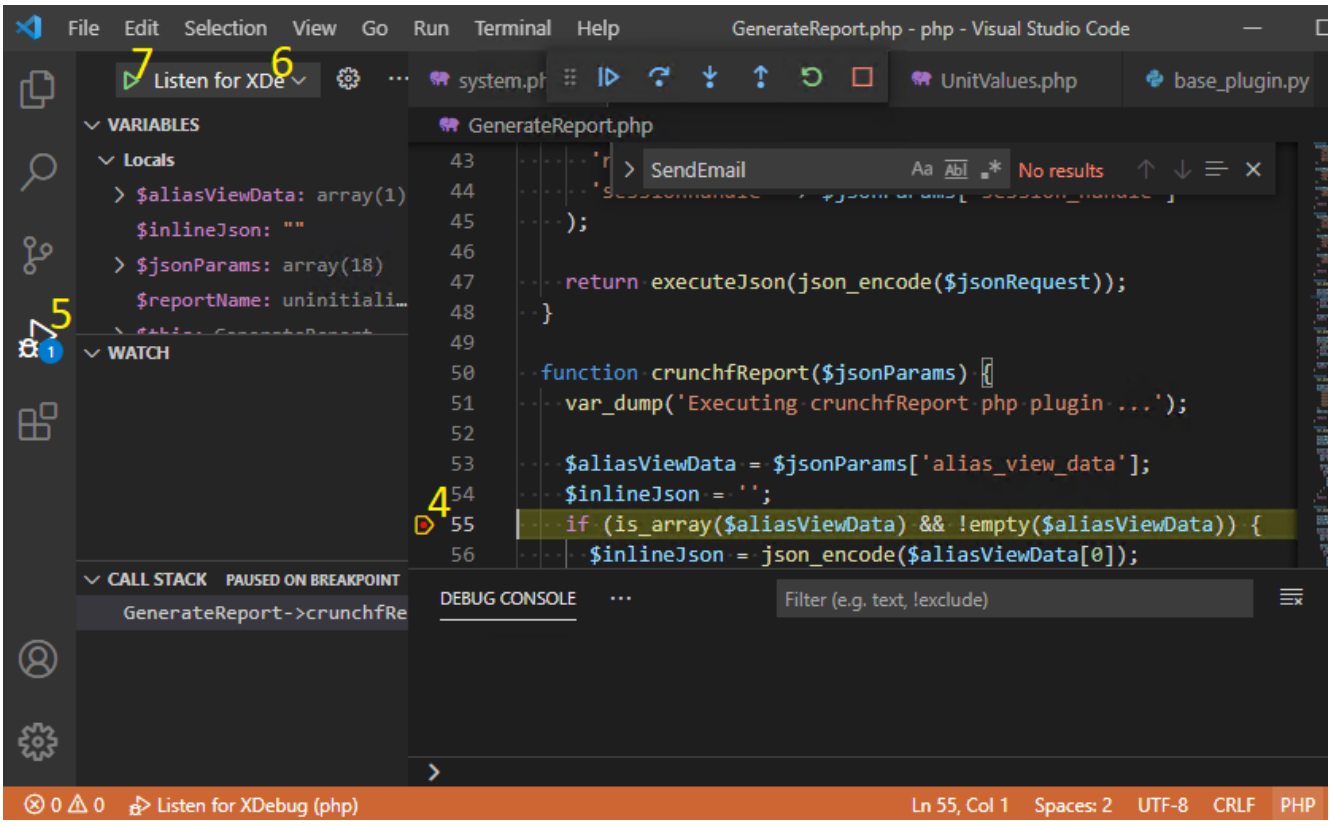
For debugging a PHP script, the following parameters should be set:

- php_xdebug.dll is in the extension list.
- xdebug.mode must be set to debug.
- client_host and client_port matched the VS Code launch.json file.

8.3 Debugging

1. Run the service in administrator account.
2. Run VS Code in administrator account.
    - VS Code must be run with the same account as service.

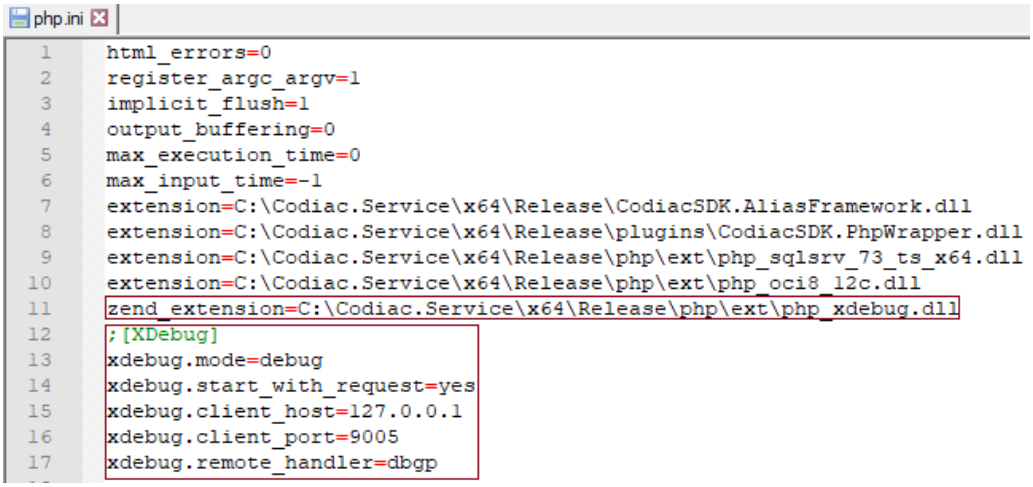
3. Open the PHP directory to be debugged.
   - AF extension function: *<service root>\Modules\php.*
   - plugins: *<service root>\plugins\php.*
4. Add a breakpoint by clicking the left side of the line number.
5. Click the debug icon on the left side bar.
6. Select the **Listen for XDebug**.
7. Click the green play button (or F5) to start debugging. At this point, VS Code is attached to the debugger.
8. Execute the **PHP plugin** or **PHP AF extension function**.
9. Wait for a few seconds to stop at breakpoint.



10. You can step through the codes by pressing the F10 (step next), F11 (step into), and F5 (continue) keys or stop (orange square icon). You can add watches or see local watches.

9 Remote Debugging PHP plugin and AF extension functions

In the PHP language, remote debugging is almost identical to local debugging.
To set the remote debugging PHP plugin and AF extension functions, configure the **php.ini** file which is located in service root directory. Below are the default settings:



For debugging a PHP script, the following parameters should be set:

- php_xdebug.dll is in the extension list.
- xdebug.mode must be set to debug.
- client_host and client_port matched the VS Code launch.json file.

The IP address (client_host) and the port (client_port) in the **php.ini** file should be visible and accessible from the outer machine.
Then, perform the debugging steps as for the local debugging described above in the Debugging section.

10 Debugging C# plugin and AF extension functions

10.1 Required tools

Download and install the tools below:

- **Visual Studio 2019**
  The Visual Studio 2019 tool can be downloaded at the following link:
  https://visualstudio.microsoft.com/downloads
  - Install **.Net desktop development**.
  - Install the **Just-In-Time debugger**.

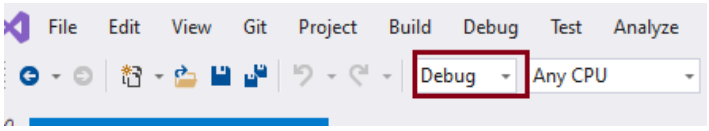
10.2 Tool configurations

10.2.1 Project configuration

On Visual Studio Solution Explorer, right click on the project, click Properties, and select Application to view the project properties. At the moment, the C# project must be built with Net Framework 4.5.2.

10.2.2 Build C# project with debug symbols

Select the Debug configuration and build C# project (Ctrl-Shift-B). Then, copy the output DLL files to *<service root>\Debug\Modules\net* directory for AF extension module; or *<service root>\Debug\plugins\net* for plugin.

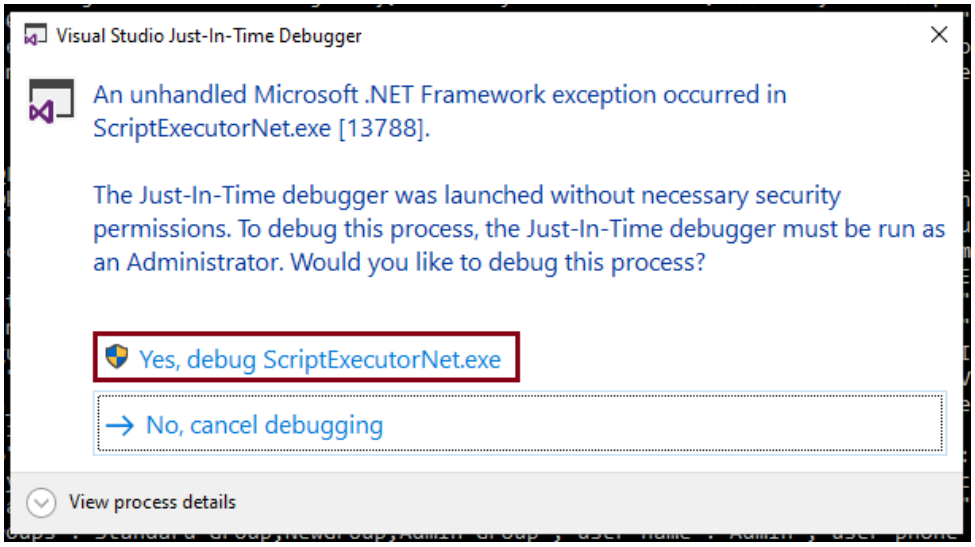


10.3 Debugging

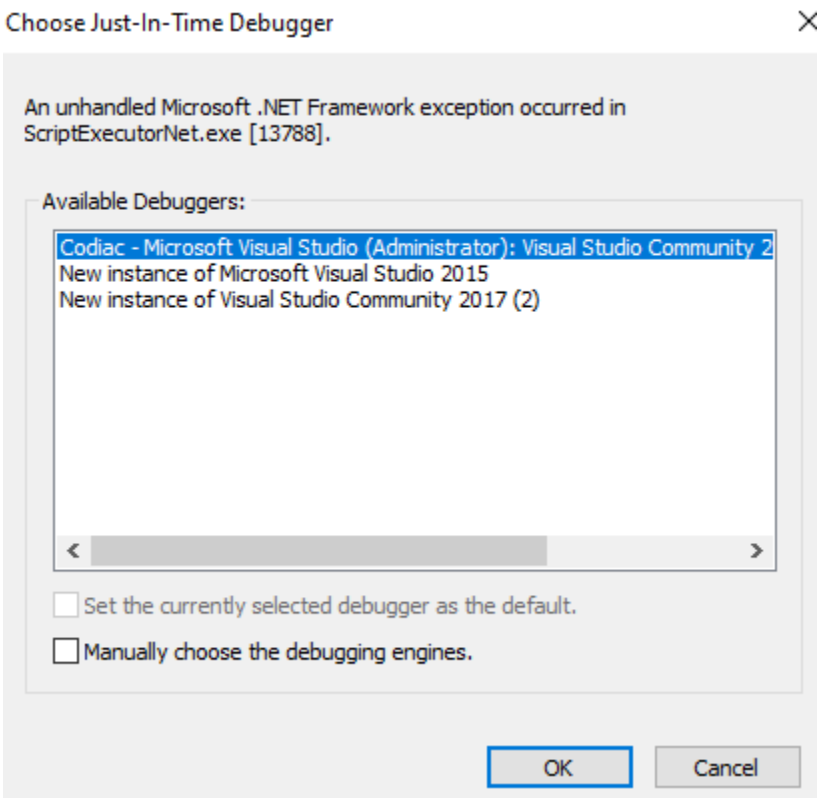
1. Insert `System.Diagnostics.Debugger.Launch();` to where you want to start the debug. This will invoke debugger at runtime.

```
[Method("example plugin to run SQL select query")]
public JObject SelectCrunch(JObject jsonParams)
{
    System.Diagnostics.Debugger.Launch();

    // Select SQL syntax is:
    // SELECT "table name"."Column name" FROM "table name"

    string sqlQuery =
    " SELECT \"Crunch\".\"Contract\"," +
    "    \"Crunch\".\"Owner\" AS \"Person\"," +
    "    \"Crunch\".\"MaturityDate\" AS \"Date\"," +
    "    \"CrunchF\".\"Amt\" AS \"Amount\" " +
```

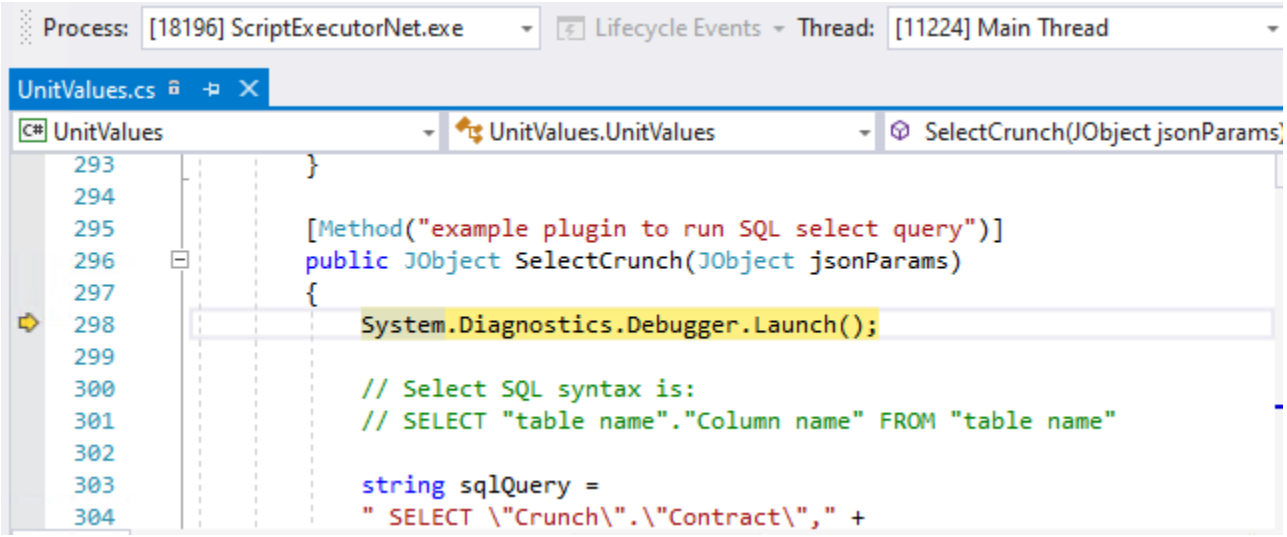
2. Build plugin or module (with Debug symbols) and copy the output DLL files to *<service root directory>\plugins\net* directory (for a plugin) or *<service root directory>\Modules\net* directory (for a module function).
3. Run the service in administrator account.
4. Execute the plugin or module function. This will prompt a Just-In-Time debugger window. Click the **Yes** button.



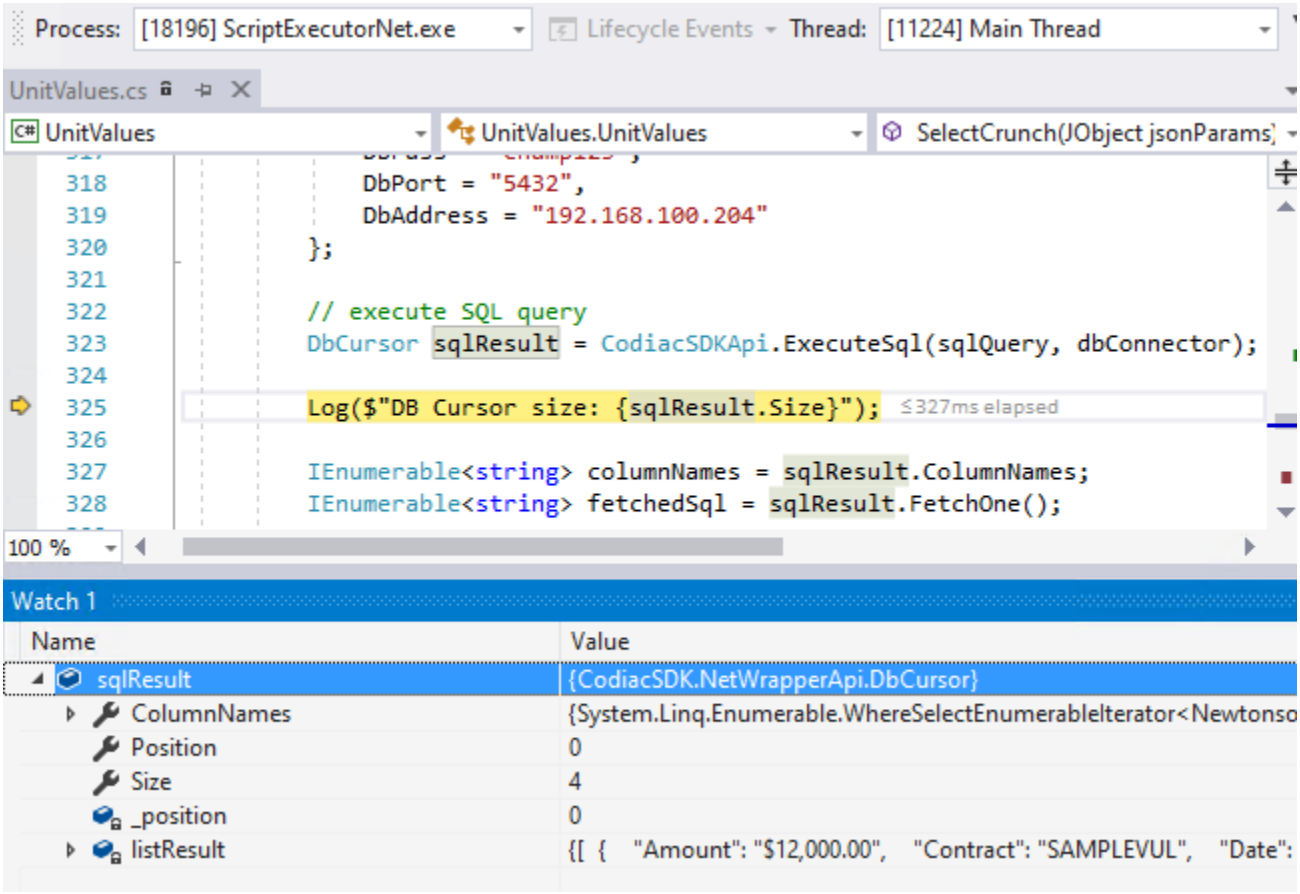
5. Select a Visual Studio debugger instance.



6. At this point, the debugger should stop at **Debugger.Launch()**.



7. You can step through the code by pressing the F10 (step over) and F11 (step into) keys and see the variable values in watches.



8. To end debugging, open the **Debug** menu and select the **Detach all** menu item.

11 Remote Debugging C# plugin and AF extension functions

Full instruction for the remote debugging for the C# applications can be found on the Remote Debug a C++ Project - Visual Studio (Windows) | Microsoft Learn page.

11.1 Required tools

Download and install the tools below on the remote machine:

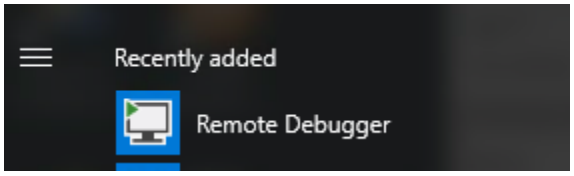
- **Remote Debugger**
  The Remote Debugger can be downloaded at the following link: Remote Debug a C++ Project - Visual Studio (Windows) | Microsoft Learn

11.2 Tool configurations

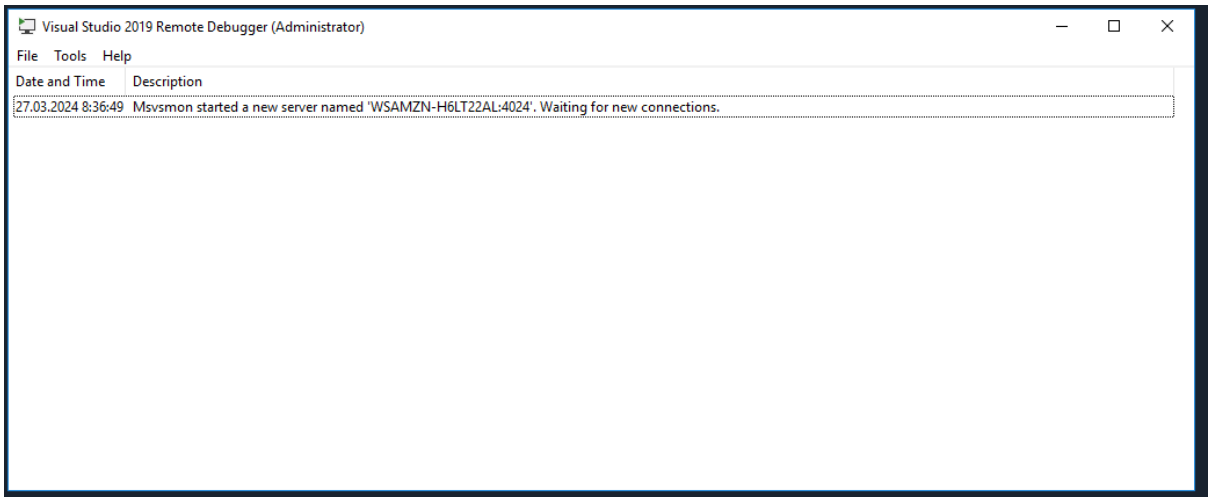
11.2.1 Install Remote Debugger

To set up the tool for remote debugging, follow the steps below:

1. Download the Remote Debugger.
2. Install the Remote Debugger on the remote machine.
3. Launch the Remote Debugger on the remote machine.

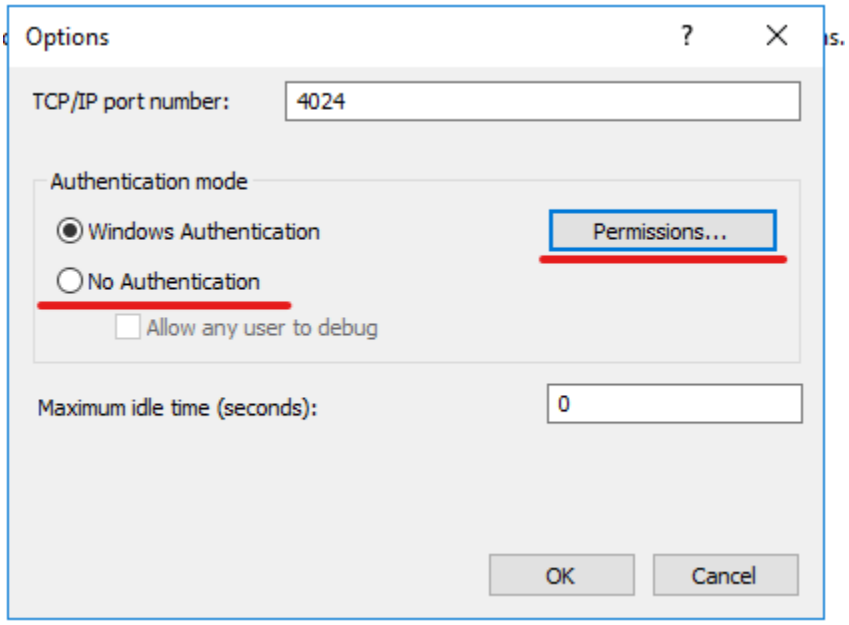


The Remote Debugger working screen will be displayed as follows:



In the Remote Debugger working screen, perform the following actions, to configure the access to the Remote Debugger:

1. In the Menu, click the **Tools** menu section and select the **Options…** menu item. The Options pop-up window opens.



In the Options pop-up window, fill in the necessary fields:

- **TCP/IP port number** – the communication endpoints that applications use to coordinate and establish communication over a network using the TCP/IP protocol suite.

Based on the desired security level select:

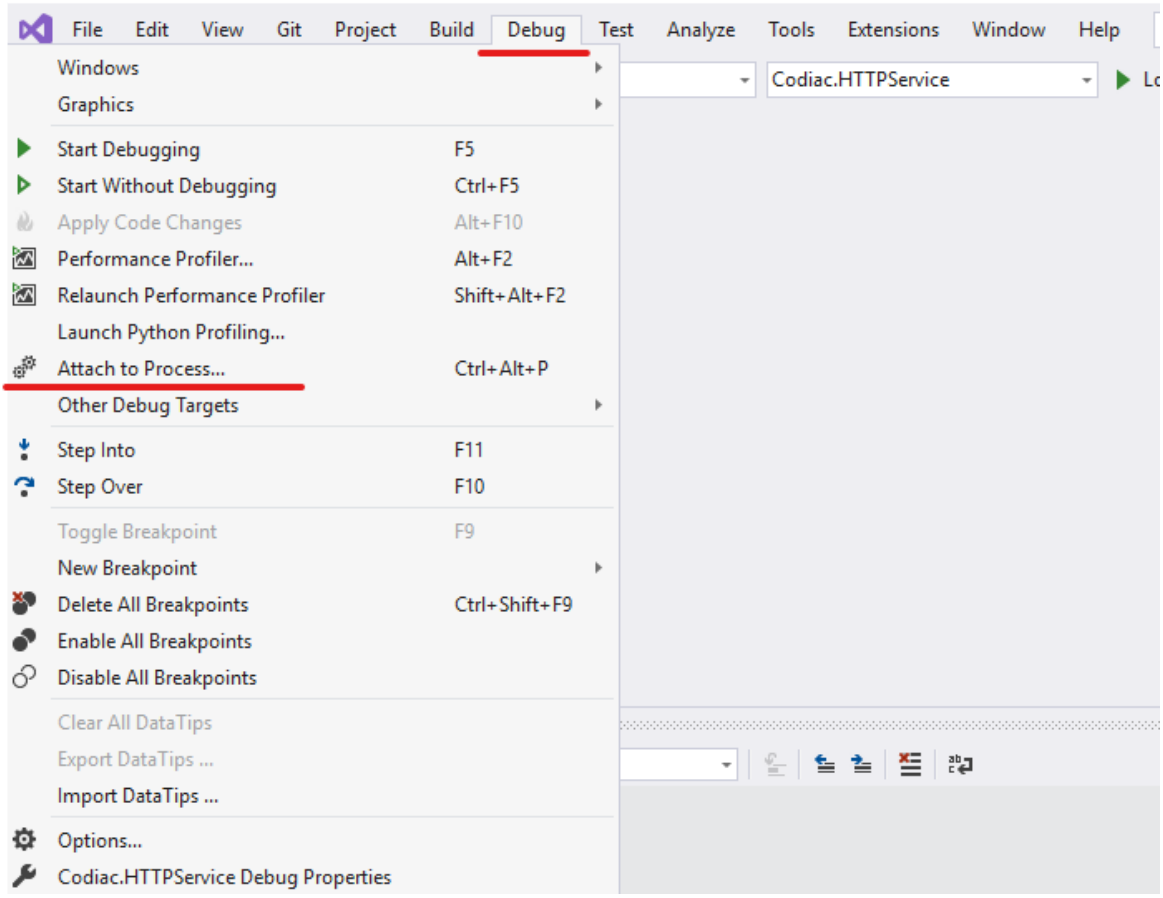
- the **Windows Authentication** access and set the **Permissions** to a user,
  or
- the **No Authentication** access for the remote debugging.

After the above-mentioned options are set, the remote machine will be ready to accept the Remote Debugger's requests.

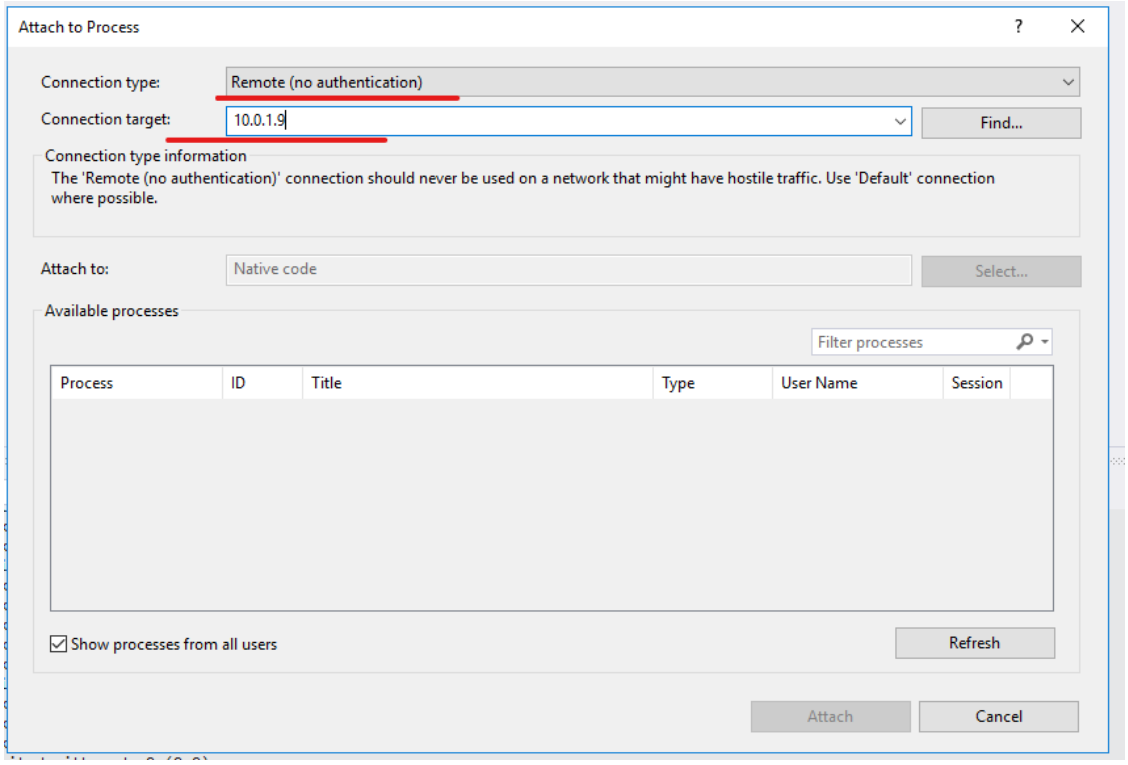
11.3 Debugging

On the local machine where plugins or modules were created, users just need to select the process in the Visual Studio. The Remote Debugger will be attached to the selected process. For that:

1. Launch Visual Studio.
2. Click the **Debug** section on the menu, and select the **Attach to Process…** menu item.



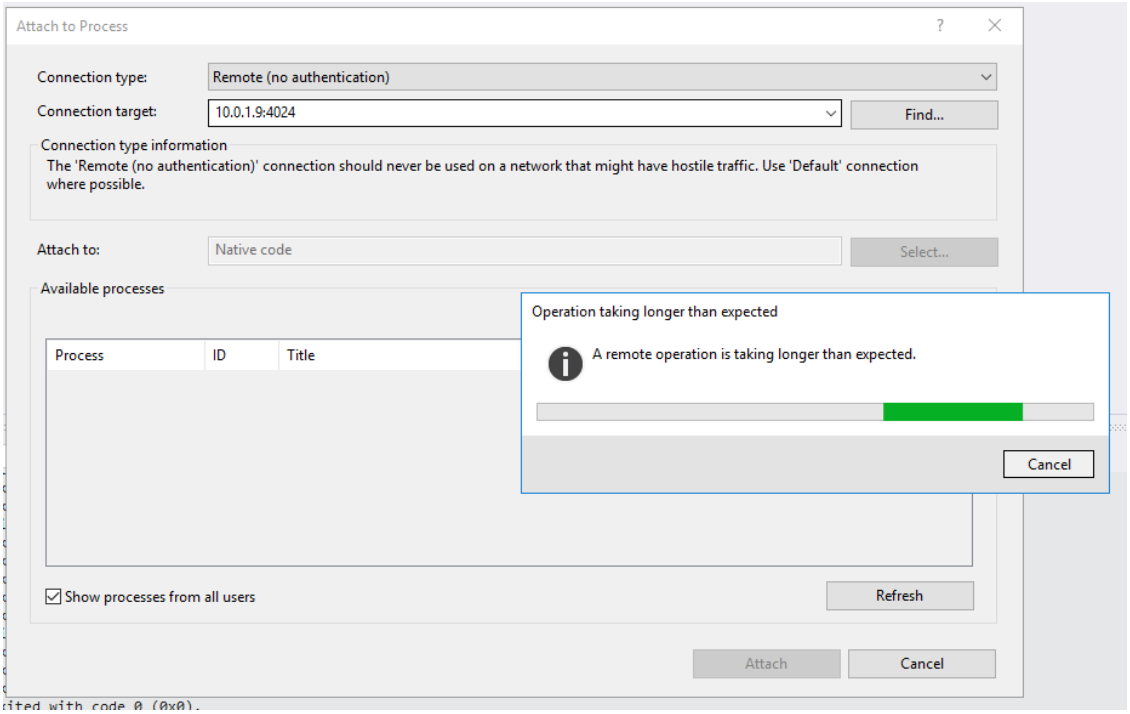
The **Attach to Process** pop-up window opens.



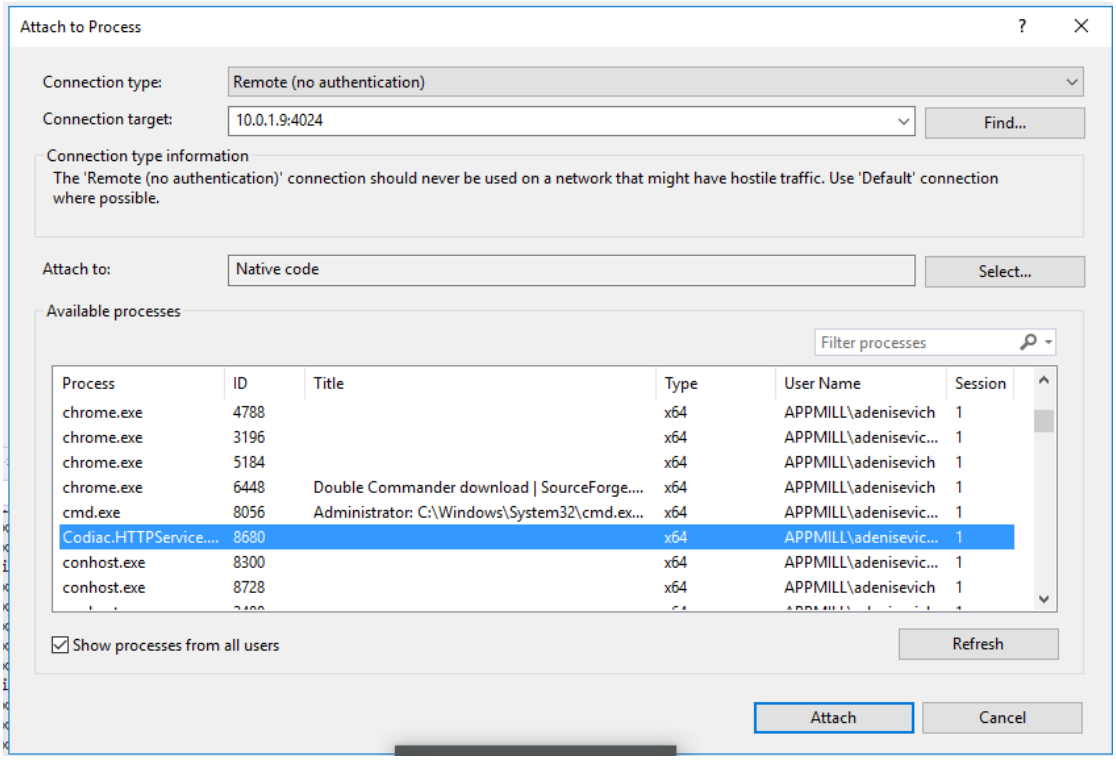
3. In the opened **Attach to Process** pop-up window, select:

- the **Remote (No authentication)** debugger type from the Connection type drop-down list,
- the remote machine IP address from the Connection target drop-down list, and
- press the ENTER key.

The debugger will connect to the remote machine, and the list of available processes that the debugger can be attached to will be shown.



4. In the available processes list, select the **Codiac.HTTPService.exe** process, and click the **Attach** button.



At reaching a code for the plugins, the process will be stopped at the breakpoint that was set in the developed plugin or modules.

To allow the remote debugging, files with the Debug information that were generated at the build should be added.