Debugging Codabix Scripts

A Step-by-Step Guide

New to Node.js and Debugging Codabix Scripts?

This guide gets you started with debugging Codabix scripts using your preferred code editor, even if you are new to Node.js and npm packages. We will walk you through setting up your local environment, creating a new project, and installing the necessary tools.

What you will need:

- **Node.js and npm:** These are essential for installing Codabix Debugging Runtime. You can download Node.js here: https://nodejs.org/en, what will also install npm. You can find detailed download instructions here: https://docs.npmjs.com/downloading-and-installing-node-js-and-npm#using-a-node-installer-to-install-nodejs-and-npm
- **Code editor:** This is where you will write and edit your code. We will use Visual Studio Code in this guide, but feel free to use your preferred editor.
- **TypeScript:** This is the programming language used for both Codabix Scripting and the code examples in this guide.
- **CDR Installer (optional):** To simplify your project setup, we provided an Installer in our GitHub Repository, which will automatically run all necessary commands – provided that Node.js is already installed on your system!
  Simply download and unpack `CDR Installer.zip` in your project folder and run the install.cmd it.
  The installer will also make necessary changes to the `tsconfig.json` and create the `package.json` required for debugging.

Setting Up Your Project:

1. **Open your project folder:** Navigate to your desired project folder in VS Code or use the terminal ("View → Terminal")
2. **Create a `package.json` file:** This file stores project information and scripts. Run `npm init -y` in the terminal to quickly create it with default settings.
3. **Install TypeScript:** Install TypeScript as a development dependency using:

```
npm install typescript --save-dev
```

4. **Create a `tsconfig.json` file:** This configures the TypeScript compiler. Run `npx tsc --init` to create it with basic settings. Open `tsconfig.json` and uncomment the `"sourceMap": true` line.
5. **Create your script:** Make a new folder (e.g., "src") and create a new TypeScript file inside it (e.g., "testScript.ts") for your script.

6. **Configure debugging (VS Code specific):** Go to “Run and Debug” (Ctrl+Shift+D) and select “create a launch.json file”. Choose Node.js as debugger. Edit the file path to your .ts file as needed.
   Edit `launch.json` and add the following line to the “configurations” section:

   ```
   "preLaunchTask": "tsc: build - tsconfig.json",
   ```

   This allows automatic compilation of TypeScript code to JavaScript after making changes, saving you manual compilation steps whenever you press F5 to debug. Alternatively, you can run `npx tsc` manually each time before debugging your script.

   Here is an example:

   ```
   {
     "configurations": [
       {
         "type": "node",
         "request": "launch",
         "name": "Launch Program",
         "program": "${workspaceFolder}/src/testScript.ts",
         "preLaunchTask": "tsc: build - tsconfig.json",
         "outFiles": ["${workspaceFolder}/**/*.js"]
       }
     ]
   }
   ```

7. **Install Codabix Debugging Runtime:** Finally, install the debugging package with its dependencies:

   ```
   npm install @traeger-instustry/codabix-debug-runtime
   ```

Note: This guide provides a basic overview. Refer to the official documentation for advanced setup and configuration options.

Remember:

- This guide uses VS Code for demonstration purposes, but feel free to use your preferred editor.
- You can find more information and resources for Node.js, npm, and TypeScript in their respective documentation.

Debugging a Script

Importing Modules

Unlike Codabix script development, where modules are automatically available, you need to explicitly import them for use in your script.

The following modules are available for use in your script: codabix, guid, io, logger, net, runtime, timer.

To use one or more of these modules, you need to import them at the beginning of your script file (e.g., testScript.ts). Here is an example:

```
import { codabix, logger } from "@traeger-industry/codabix-debug-runtime";
```

This example imports the `codabix` and `logger` modules from the package.

Note: You only need to import the modules you use in your script.

Using a Codabix Node Model File

By default, this package relies on a file named “codabix-nodes.xml” located in your project directory. If this file is missing, any actions related to nodes will not work.

Creating the Codabix Node Model File

1. **Open an existing Codabix Project.** This can be a project you have already created or a new one.
2. **Go to the node explorer.**
3. **Right-click on the topmost node.** This is the main node in your project.
4. **Select “Export Nodes as XML” from the menu.** This will create a new file named “codabix-nodes.xml”. Save this file to your project directory.

Changing the File Path (Optional)

By default, the package looks for the file named “codabix-nodes.xml”. If you want to use a different file or location, you can change the path using code like this:

```
// Change the path to your file (e.g., "./my-nodes.xml")
codabix.Debug.model = "./my-nodes.xml";
```

Important Note: Modifying the nodes directly in your project (adding, deleting, or editing them) will not affect the contents of the “codabix-nodes.xml” file.

Optional Runtime Setup

This package allows you to optionally configure the runtime environment to simulate different scenarios while developing and testing your code. These scenarios can represent situations like limited resources or communication delays.

Node Value Status

In Codabix, when a node provides a value, it also includes a “status” that indicates whether the value was obtained successfully. You can use the `codabix.Debug.defaultStatus` property to control the default status for simulated values during development:

- **Default (true):** The simulated node values will have a good status (`codabix.NodeStatusValueEnum.Good`), indicating successful retrieval.

- **Change to false:** All simulated node values will have a bad status (`codabix.NodeStatusValueEnum.Bad`), as if there were issues obtaining the data.

Code example:

```
// Set all simulated node values to have bad status
codabix.Debug.defaultStatus = false;
```

Communication Delays

In real-world Codabix projects, communication with other components might have delays due to resource limitations. This package allows you to simulate these delays during development:

- **Reading data:** Use the `codabix.Debug.readDelay` property to set a delay (in milliseconds) for simulated read operations using `codabix.readNodeValueAsync()`.
- **Writing data:** Use the `codabix.Debug.writeDelay` property to set a delay (in milliseconds) for simulated write operations using `codabix.writeNodeValueAsync()`.

Code examples:

```
// Simulate a 2-second delay for reading data. Default value: 1000.
codabix.Debug.readDelay = 2000;

// Simulate a half-second delay for writing data. Default value: 1000.
codabix.Debug.writeDelay = 500;
```

Example: Putting It All Together

Here is an example of a simple script that writes a value to a node:

```
import { codabix, logger, runtime } from "@traeger-industry/codabix-debug-
runtime";

codabix.Debug.model = "./codabix-nodes.xml";
codabix.Debug.writeDelay = 100;

runtime.handleAsync(async function () {
    // This is the main code.
    const nodeIdentifier = "/Nodes/FolderC/FolderCC/VarCC1";
    await writeValue(nodeIdentifier, 2);

    logger.log(`Wrote value to node '${nodeIdentifier}',` +
        `value: ${ codabix.findNode(nodeIdentifier, true).value }.`);

    async function writeValue(nodeIdentifier: string, increment: number) {
        // Find the specified node and write a value to it.
        const node = codabix.findNode(nodeIdentifier, true);
        let nodevalue: number = Number(node.value)
```

```
        while (Number(node.value) !== Number(node.maxValue)) {
            nodevalue = Number(node.value);
            await codabix.writeNodeValueAsync(node, (nodevalue + increment));
            logger.log(`Current node value: '${node.value}'`);
        }
    }
}());
```

This is the log from the Debug Console:

```
C:\Tools\nodejs\node.exe .\src\testScript.js
[Info] Current node value: '4'
[Info] Current node value: '6'
[Info] Current node value: '8'
[Info] Current node value: '10'
[Info] Current node value: '12'
[Info] Current node value: '14'
[Info] Current node value: '16'
[Info] Current node value: '18'
[Info] Current node value: '20'
[Info] Current node value: '22'
[Info] Current node value: '24'
[Info] Current node value: '26'
[Info] Wrote value to node '/Nodes/FolderC/FolderCC/VarCC1',value: 26.
```

Getting More Help

This guide equips you with the basics of setting up your environment and debugging Codabix Scripts. If you encounter any issues or want to delve deeper into specific topics, here are some helpful resources:

- **Codabix Script Interface Plugin Development Guide:** https://www.codabix.com/en/plugins/interface/scriptinterfaceplugin/scriptinterface.development.guide
- **Node.js Documentation:** https://nodejs.org/en
- **npm Documentation:** https://docs.npmjs.com/

For any further questions or issues, please do not hesitate to contact Codabix Support at support@traeger.de.