



Tutorial A7: Debugging a smart contract


Estimated time: **15 minutes**

In the last tutorial we upgraded our smart contract by adding a new transaction type to our smart contract called 'queryAllAssets'. In this tutorial we will:

- Understand the tools available in IBM Blockchain Platform to debug smart contracts
- Use the VS Code debugger to step through our new transaction and see how it works
- Use watches to see how we can monitor variables in smart contracts


This tutorial is not intended to demonstrate all the features of the VS Code debugger. For more information, see the [VS Code debugger documentation](#).

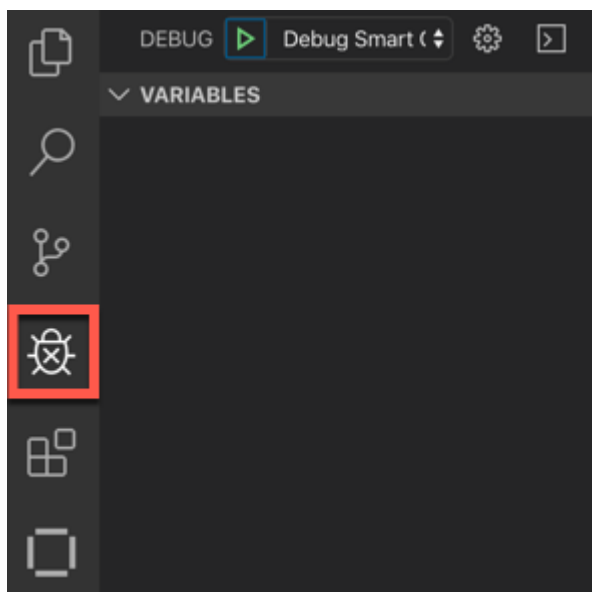
In order to successfully complete this tutorial, you must have first completed tutorial [A6: Upgrading a smart contract](#) in the active workspace.

 **A7.1:** Expand the first section below to get started.

► Start a debug session

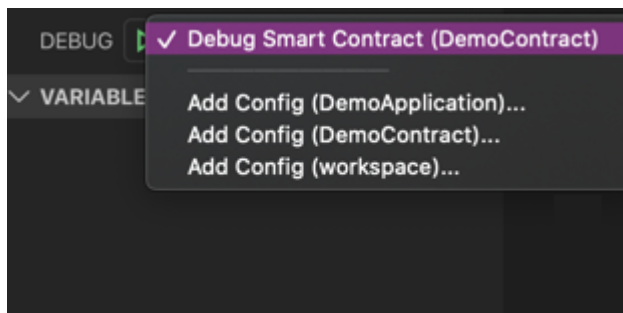
The VS Code debugger contains four views in its own sidebar: Variables, Watch, Call Stack and Breakpoints.

 **A7.2:** Click the Debugger icon to show the Debugger sidebar.



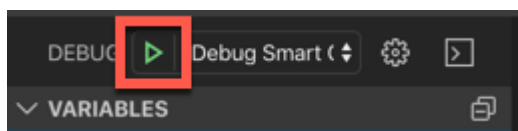
At the top of the sidebar is a dropdown list that shows the available debug configurations. IBM Blockchain Platform will have added a configuration to allow us to debug our DemoContract smart contracts.

- A7.3: Click the drop-down list at the top of the Debugger sidebar and ensure that 'Debug Smart Contract (DemoContract)' is selected.



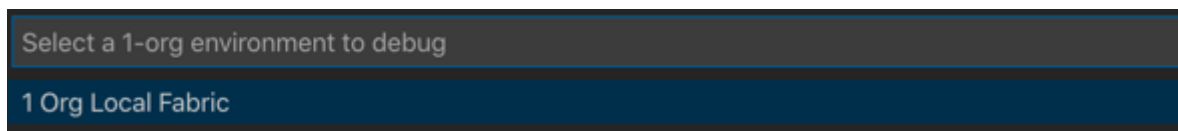
We will now start a session to debug our DemoContract smart contract.

- A7.4: Click the green Start arrow to the left of the drop-down list.

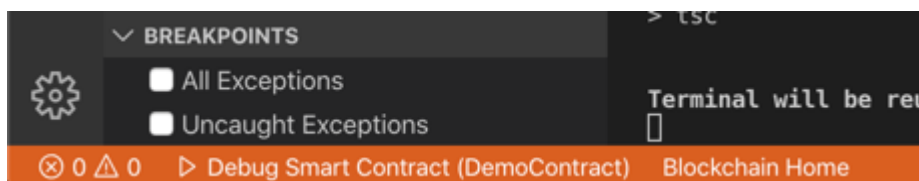


We now need to tell VS Code the location of the Fabric Environment in which we are going to debug our smart contract.

- A7.5: In the command palette that appears, click '1 Org Local Fabric'.



After a brief pause, the bar at the bottom of VS Code will change color to indicate that a debug session has started.



If necessary, VS Code will prompt us to upgrade the smart contract at this time. If you are prompted to do this, just accept any defaults and let the upgrade complete.

- A7.6: Expand the next section of the tutorial to continue.

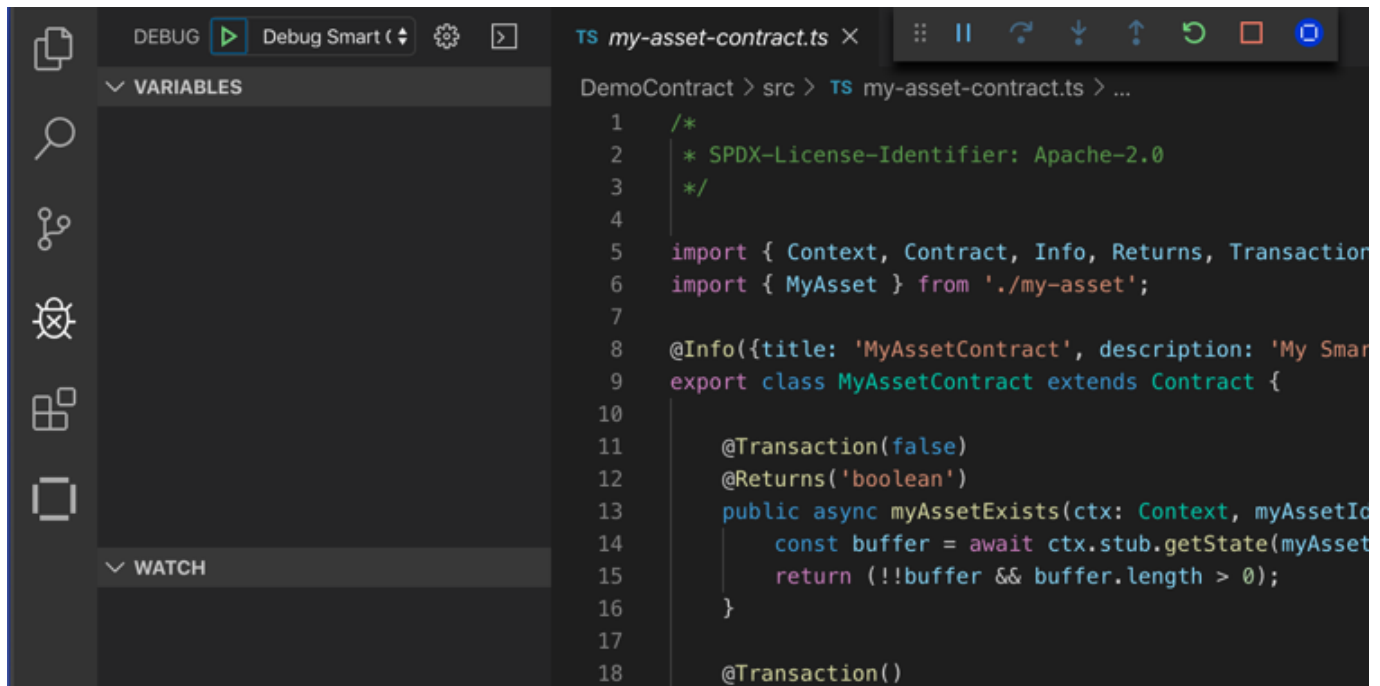
► Step through a transaction

We are now going to run our new *queryAllAssets* transaction in the debugger to see how it works. We will set a breakpoint in the smart contract to allow us to pause and step through the code.

To do this, we first need to ensure that our smart contract has focus in the editor.

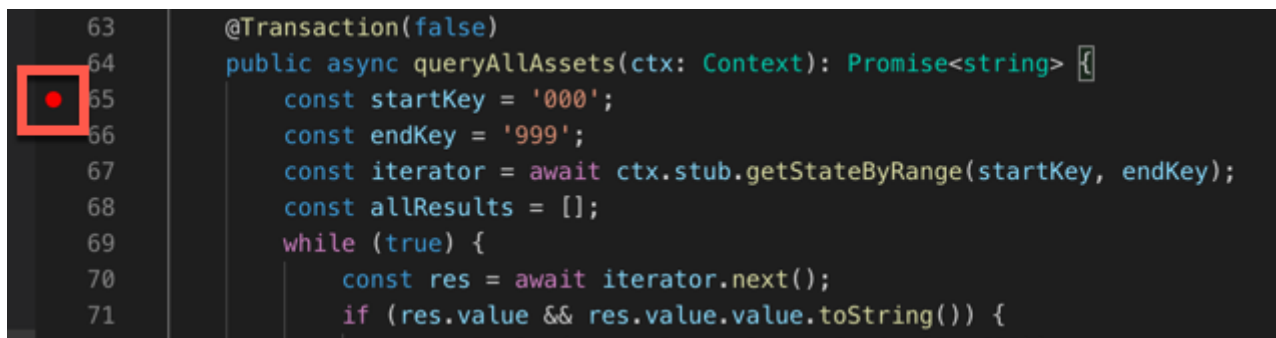
A7.7: Click on the *my-asset-contract.ts* tab in the editor.

If the file is not loaded in the editor, you will need to click the Explorer sidebar, load the DemoContract -> src -> my-asset-contract.ts file, then click back to the Debugger sidebar.



We will now set a breakpoint.

A7.8: Scroll to the first statement of the *queryAllAssets* method and click the mouse just to the left of the line number.



As with all debuggers, this causes execution to pause whenever this statement is reached.

We will now invoke a transaction to cause this method to be called.

A7.9: Click the blue IBM Blockchain Platform icon in the Debug bar at the top of the screen.



A7.10: Click 'Evaluate Transaction'.

Choose a command to execute

Submit Transaction

Evaluate Transaction

A7.11: Click 'MyAssetContract - queryAllAssets'.

Choose a transaction to evaluate

MyAssetContract - myAssetExists

MyAssetContract - createMyAsset

MyAssetContract - readMyAsset

MyAssetContract - updateMyAsset

MyAssetContract - deleteMyAsset

MyAssetContract - queryAllAssets

As you will recall, there are no arguments or transient data to supply to this transaction.

A7.12: Press Enter to specify no arguments on the transaction.

|

optional: What are the arguments to the transaction, (e.g. ["arg1", "arg2"]) (Press 'Enter' to confirm or 'Escape' to cancel)

A7.13: Press Enter to specify no transient data.

|

optional: What is the transient data for the transaction, e.g. {"key": "value"} (Press 'Enter' to confirm or 'Escape' to cancel)

The transaction will now start to run, but will pause at the breakpoint we set.

```
62
63     @Transaction(false)
64     public async queryAllAssets(ctx: Context): Promise<string> {
65         const startKey = '000';
66         const endKey = '999';
67         const iterator = await ctx.stub.getStateByRange(startKey, endKey);
68         const allResults = [];
69         while (true) {
70             const res = await iterator.next();
71             if (res.value && res.value.value.toString()) {
72                 console.log(res.value.value.toString('utf8'));
73             }
74         }
75     }
```

A7.14: Click 'Step Over' in the Debug bar multiple times to progress through the transaction's implementation.



As you step through, note that the Variables and Call Stack views change depending on the current scope.

You should also see that the *while* loop is called twice before finishing; this is because, as you may recall, there are two assets in the world state ('002' and '003').

Transaction timeouts?

You might see a transaction timeout error while the debugger is paused. This is OK, and you can use the debugger to explore how errors are handled. Click 'Continue' in the Debug bar and invoke the transaction again if you wish.

- **A7.15:** When you have stepped through to the end of the transaction, click 'Continue' to unpause execution.



- **A7.16:** Expand the next section of the tutorial to continue.

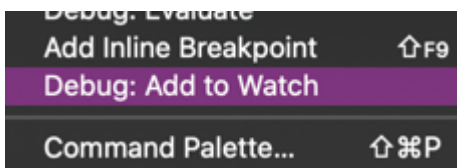
► Add a variable watch

As with other debuggers, it is also possible to put watches on variables and expressions so that you can check if and when certain conditions hold.

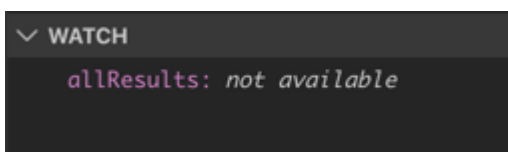
- **A7.17:** Select the first occurrence of *allResults* in the *queryAllAssets* method, where this variable is initialized.

```
66      const endKey = '999';
67      const iterator = await ctx.st
68      const allResults = [];
69      while (true) {
70          const res = await iterator
```

- **A7.18:** Right click over the selected text and select 'Debug: Add to Watch'.



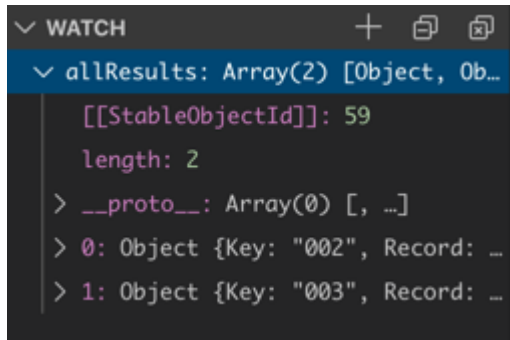
You will now see 'allResults' appear in the Watch view.



- **A7.19:** Click the blue IBM Blockchain Platform Debug icon in the debug bar a second time, and repeat the earlier steps to evaluate the *queryAllAssets* transaction again.

As you step through the debugger, look particularly at the Watch view to see how the *allResults* array is built up as the transaction progresses. Note that watch variable is shown as a tree, which you can expand for more

details.



```
WATCH
  allResults: Array(2) [Object, Ob...
    [[StableObjectId]]: 59
    length: 2
  > __proto__: Array(0) [, ...]
  > 0: Object {Key: "002", Record: ...}
  > 1: Object {Key: "003", Record: ...}
```

Try setting additional breakpoints on other transactions and running them through the debugger.

 **A7.20:** When you are finished, click the Stop button in the debug bar to stop the debug session.



Hot fixes

It is not possible to make changes to smart contracts while debugging. You must stop the debugger before your smart contract can be upgraded.

Summary

In this tutorial we have used the debugger that is built into VS Code to step through a smart contract that is deployed to our Fabric environment.

In the next tutorial we will see how we can generate functional tests for our smart contract's transactions.

→ **A8: Testing a smart contract**