



Tutorial A4: Invoking a smart contract from VS Code

Estimated time: 15 minutes

In the last tutorial we packaged, installed and instantiated our smart contract on a locally defined instance of Hyperledger Fabric. In this tutorial we will:

- Learn about gateways and wallets
- Understand the different types of transactions
- Submit transactions that call the smart contract directly in VS Code

In order to successfully complete this tutorial, you must have first completed tutorial [A3: Deploying a smart contract](#) in the active VS Code workspace.

 **A4.1:** Expand the first section below to get started.

► Connect to the Hyperledger Fabric gateway

In order to submit transactions in Hyperledger Fabric you will need to use a *gateway*.

Fabric Gateways and Wallets

A gateway represents a connection to a Hyperledger Fabric network. Applications that call transactions, whether that's VS Code or your own application, use gateways to encapsulate the details of how and where they're connecting.

Each gateway is built from within the client application and connects to a single peer on the network using a supplied identity. The credentials used to prove your identity are stored in *wallets* and passed to the gateway at connection time.

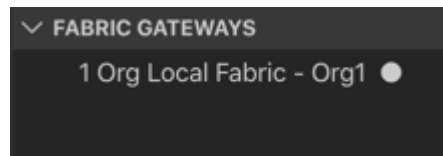
Want to know more about gateways? [Read about them in the Hyperledger Fabric documentation.](#)

Take care to understand the difference between **Fabric Environments** and **Fabric Gateways**: While a Fabric *Environment* gives an overview of all the resources available to you in a Hyperledger Fabric network, a Fabric *Gateway* represents a single connection to it. Think of a view of the entire internet (Environment), compared to a view of a single connection to a web server (Gateway).

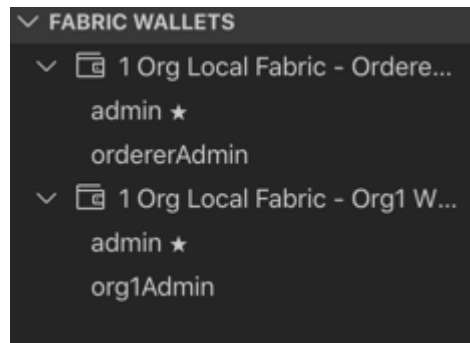
Gateways and Wallets in VS Code

The gateways that have been configured in the VS Code workspace are shown in the Fabric Gateways view in the IBM Blockchain Platform extension.

When the smart contract was deployed in the previous tutorial, a gateway was created for you at the same time; this is now shown in the Fabric Gateways view.



Furthermore, the available wallets and identities are shown in the Fabric Wallets view:



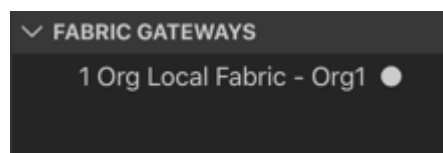
As you can see, four identities are listed in total: two each for the orderer and peer organizations that were created as part of the previous tutorial. The '*admin*' identities in each organization are responsible for running the Hyperledger Fabric component (peer or orderer). The other identities ('*ordererAdmin*' and '*org1Admin*') are intended for client connections, and is what we will use for the gateway.

Connecting to the Fabric Gateway in VS Code

We will now connect to the local peer using the Fabric Gateway instance that is available to us.

 **A4.2:** In the Fabric Gateways view, click "1 Org Local Fabric - Org1".

If you can't see this view, remember to first click the IBM Blockchain Platform icon in the sidebar.



You will now be asked to select the identity that the gateway will use to connect to the network. Two are listed; these are the two identities in the wallet for the peer's organization (Org1). As a reminder, we will be using the identity *org1Admin*.

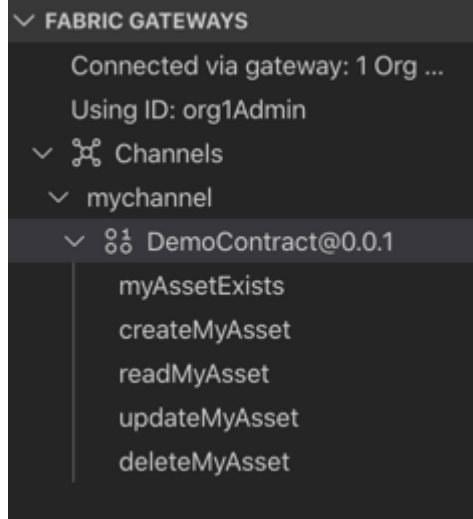
 **A4.3:** Click 'org1Admin'.



The IBM Blockchain Platform will now connect to the locally running Hyperledger Fabric instance; this will only take a few seconds to complete.

Once connected, notice that the Fabric Gateways view changes to reflect the channels and transactions available to the connected gateway.

 **A4.4:** Fully expand the Channels tree in the Fabric Gateways view to show the available transactions.



The tree shows the network into which the smart contract was deployed (mychannel), the name of the smart contract that was deployed (DemoContract@0.0.1), and the five transaction methods that were implemented as part of the smart contract.

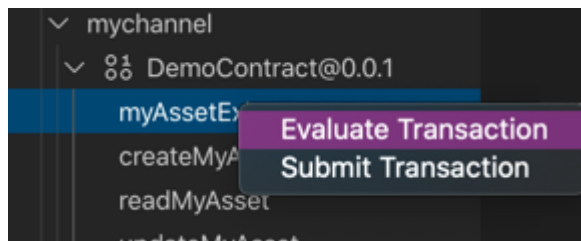
- **A4.5:** Expand the next section of the tutorial to continue.

► Invoke transactions using the gateway

We will now invoke some of these transactions. Hyperledger Fabric distinguishes between read/write transactions that are logged on the blockchain ledger (*submitted* transactions), and read-only transactions that are not recorded on the blockchain ledger (*evaluated* transactions). IBM Blockchain Platform allows us to invoke both of these transaction types.

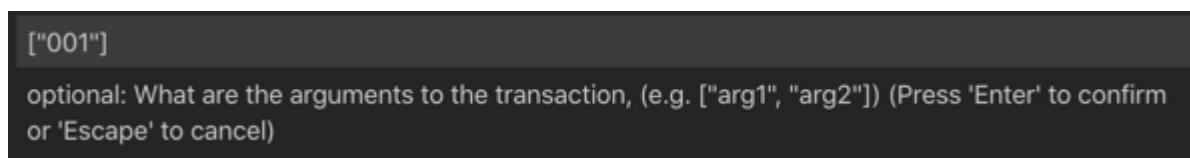
We will start by evaluating the transaction 'myAssetExists'.

- **A4.6:** Right-click 'myAssetExists' and select 'Evaluate Transaction'.



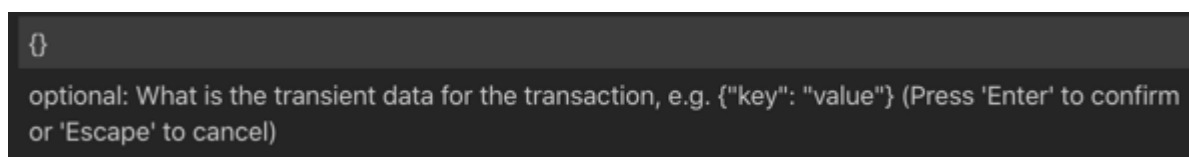
Input parameters to transactions are entered in a JSON data format.

- **A4.7:** Replace the input parameters with `["001"]` and press Enter.



Transient data is an advanced feature of Hyperledger Fabric which we will cover in a later tutorial; we will not use it here.

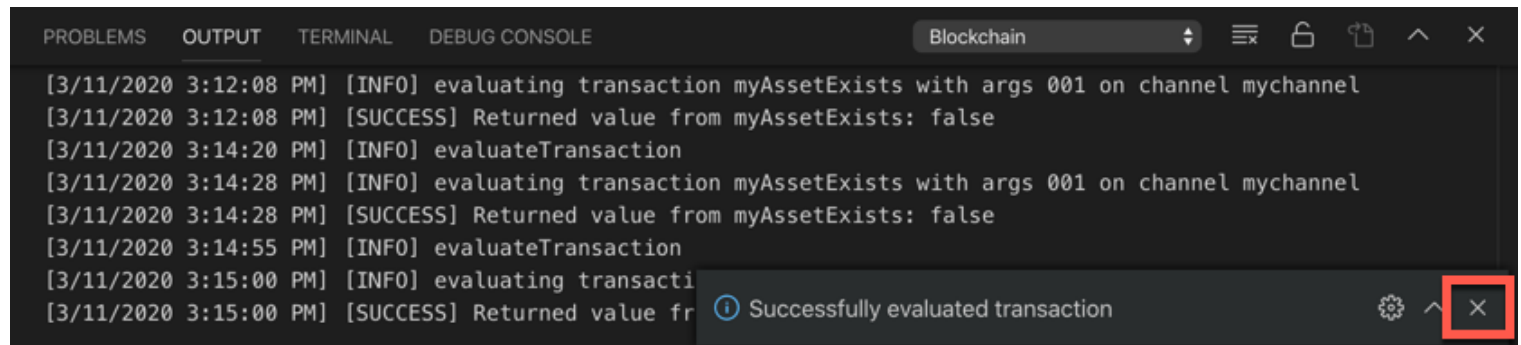
- **A4.8:** Press Enter again to accept the default transient data parameters.



The peer will now run the myAssetExists method in our smart contract using an asset ID of "001". As you will recall from the implementation, this will return true if the key "001" exists in the world state, and false otherwise.

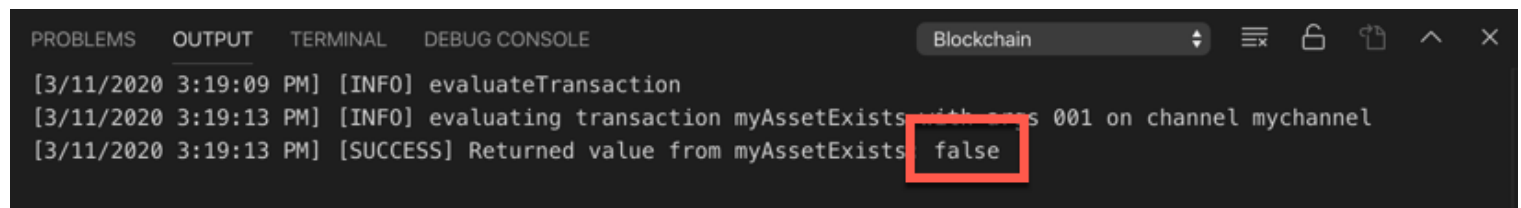
When the method completes, the Output tab will show the results of the evaluation.

- A4.9:** Move the mouse over the "Successfully evaluated transaction" notification to reveal the close icon, and click it to close it.



```
[3/11/2020 3:12:08 PM] [INFO] evaluating transaction myAssetExists with args 001 on channel mychannel
[3/11/2020 3:12:08 PM] [SUCCESS] Returned value from myAssetExists: false
[3/11/2020 3:14:20 PM] [INFO] evaluateTransaction
[3/11/2020 3:14:28 PM] [INFO] evaluating transaction myAssetExists with args 001 on channel mychannel
[3/11/2020 3:14:28 PM] [SUCCESS] Returned value from myAssetExists: false
[3/11/2020 3:14:55 PM] [INFO] evaluateTransaction
[3/11/2020 3:15:00 PM] [INFO] evaluating transaction
[3/11/2020 3:15:00 PM] [SUCCESS] Returned value from myAssetExists: false
Successfully evaluated transaction
```

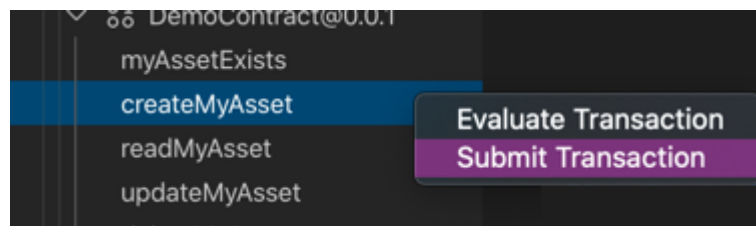
If you review the output from the command you can see the results of evaluating the transaction. In this case, the return value is false, because the key does not exist in the world state.



```
[3/11/2020 3:19:09 PM] [INFO] evaluateTransaction
[3/11/2020 3:19:13 PM] [INFO] evaluating transaction myAssetExists with args 001 on channel mychannel
[3/11/2020 3:19:13 PM] [SUCCESS] Returned value from myAssetExists: false
```

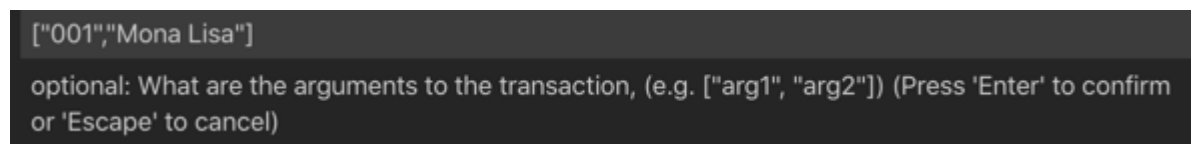
We will now create the asset. As this will modify the world state, we will need to **submit** a transaction this time rather than evaluate one.

- A4.10:** Right-click 'createMyAsset' and select 'Submit Transaction'.



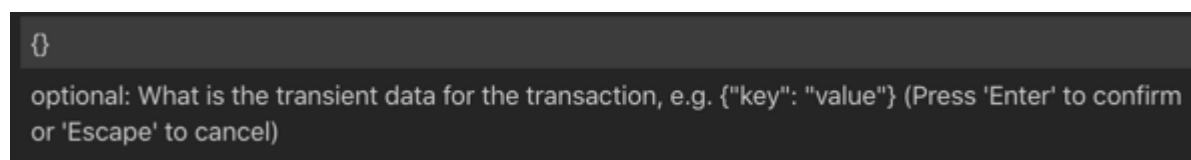
As you may recall, the createMyAsset transaction takes two parameters: a key and its associated value.

- A4.11:** Replace the input parameters with `["001", "Mona Lisa"]` and press Enter.



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

- A4.12:** Press Enter a second time to accept the transient data defaults and submit the transaction.



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

Review the output to ensure that the transaction was successful.

```
PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE Blockchain
[3/11/2020 3:19:09 PM] [INFO] evaluateTransaction
[3/11/2020 3:19:13 PM] [INFO] evaluating transaction myAssetExists with args 001 on channel mychannel
[3/11/2020 3:19:13 PM] [SUCCESS] Returned value from myAssetExists: false
[3/11/2020 3:45:22 PM] [INFO] submitTransaction
[3/11/2020 3:55:16 PM] [INFO] submitting transaction createMyAsset with args 001,Mona Lisa on channel mychannel
[3/11/2020 3:55:19 PM] [SUCCESS] No value returned from createMyAsset
Successfully submitted transaction
```

- ❑ **A4.13:** Evaluate the "myAssetExists" transaction a second time with the "001" key to show that the asset now exists.

```
[3/11/2020 4:00:58 PM] [INFO] evaluating transaction myAssetExists with args 001 on channel mychannel
[3/11/2020 4:00:58 PM] [SUCCESS] Returned value from myAssetExists: true
```

The returned value is now 'true'.

- ❑ **A4.14:** Submit the "updateMyAsset" transaction to change the value of the "001" key to "The Hay Wain".

```
[3/11/2020 4:04:46 PM] [INFO] submitting transaction updateMyAsset with args 001,The Hay Wain on channel mychannel
[3/11/2020 4:04:48 PM] [SUCCESS] No value returned from updateMyAsset
```

- ❑ **A4.15:** Evaluate the "readMyAsset" transaction to return the updated value of the "001" key.

```
[3/11/2020 4:08:09 PM] [INFO] evaluating transaction readMyAsset with args 001 on channel mychannel
[3/11/2020 4:08:10 PM] [SUCCESS] Returned value from readMyAsset: {"value":"The Hay Wain"}
```

- ❑ **A4.16:** Finally, submit the "deleteMyAsset" transaction to delete the "001" asset from the world state.

```
[3/11/2020 4:17:06 PM] [INFO] submitting transaction deleteMyAsset with args 001 on channel mychannel
[3/11/2020 4:17:08 PM] [SUCCESS] No value returned from deleteMyAsset
```

Note from this last transaction, that even though transactions in a blockchain's transaction log cannot be deleted, it is perfectly possible to delete assets from the world state. The log records the **changes** that have happened to the world state database, which can include deleting records as well as adding and modifying them.

Summary

In this tutorial we have used a gateway to connect to the Hyperledger Fabric instance running inside VS Code. We then used the IBM Blockchain Platform test capabilities to submit and evaluate transactions using the smart contract that we deployed.

In the next tutorial we will build and use a standalone application to transact with the blockchain.

→ **A5: Invoking a smart contract from an external application**