# Hands-on 2: Develop Extended HelloWorld Contract in .NET 6

In this lab, we will extend the HelloWorld program from the previous lab example, deploy it, and interact with the contract.

In the previous example, it was holding a single value of greeting. However, in this part, we will upgrade it to hold multiple greetings and cycle through them as greetings are requested.

To accomplish this, we will add a new method that allows users of the smart contract to add additional greetings.

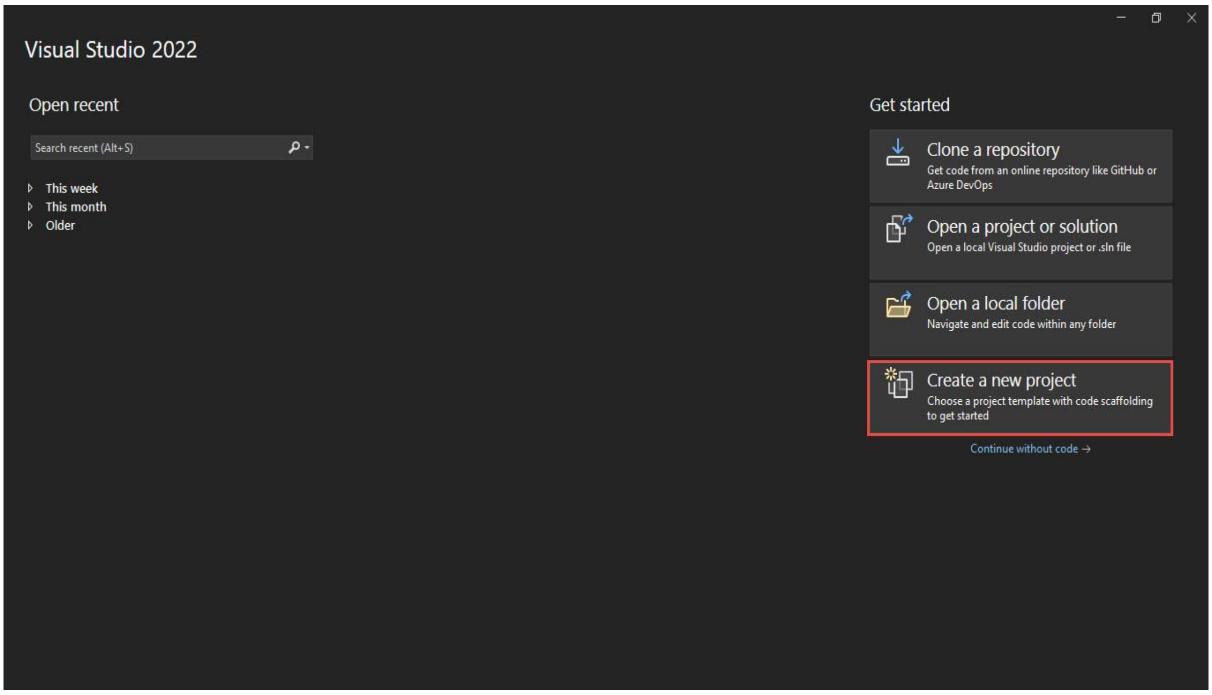
We will do the following in this lab:

- Develop the contract in .NET 6.
- Code overview
- Generate Byte code and Hash.

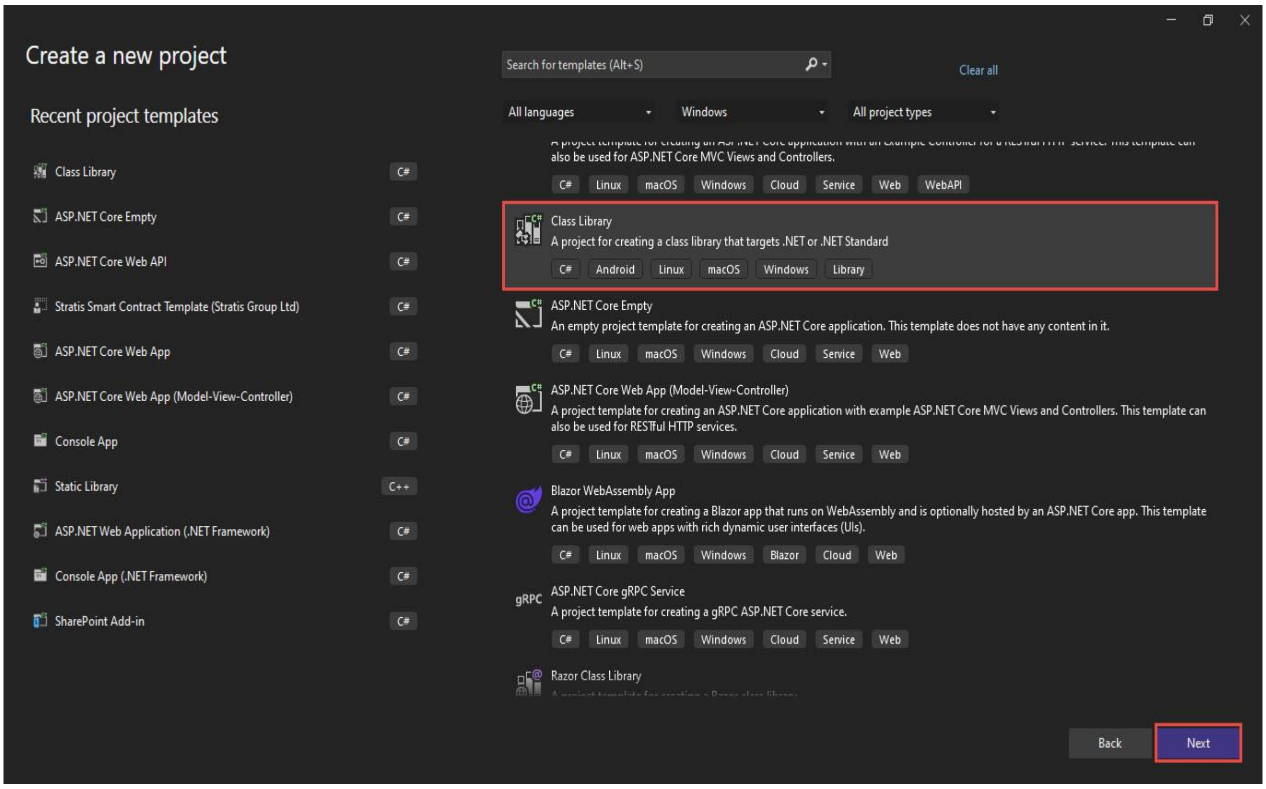
Let's first create a project.

## Create Project in Visual Studio

**Step 1-** Open Visual Studio and click on create a new project.


Visual Studio 2022
Open recent
Search recent (Alt+S)
This week
This month
Older
Get started
Clone a repository
Get code from an online repository like GitHub or Azure DevOps
Open a project or solution
Open a local Visual Studio project or .sln file
Open a local folder
Navigate and edit code within any folder
Create a new project
Choose a project template with code scaffolding to get started
Continue without code →

**Step 2**- Select class library project in C# as illustrated below and click on Next.


Create a new project

Recent project templates

| | |
|---|---|
| Class Library | C# |
| ASP.NET Core Empty | C# |
| ASP.NET Core Web API | C# |
| Stratis Smart Contract Template (Stratis Group Ltd) | C# |
| ASP.NET Core Web App | C# |
| ASP.NET Core Web App (Model-View-Controller) | C# |
| Console App | C# |
| Static Library | C++ |
| ASP.NET Web Application (.NET Framework) | C# |
| Console App (.NET Framework) | C# |
| SharePoint Add-in | C# |

Search for templates (Alt+S) | Clear all

All languages | Windows | All project types

also be used for ASP.NET Core MVC Views and Controllers.
C# Linux macOS Windows Cloud Service Web WebAPI

Class Library
A project for creating a class library that targets .NET or .NET Standard
C# Android Linux macOS Windows Library

ASP.NET Core Empty
An empty project template for creating an ASP.NET Core application. This template does not have any content in it.
C# Linux macOS Windows Cloud Service Web

ASP.NET Core Web App (Model-View-Controller)
A project template for creating an ASP.NET Core application with example ASP.NET Core MVC Views and Controllers. This template can also be used for RESTful HTTP services.
C# Linux macOS Windows Cloud Service Web

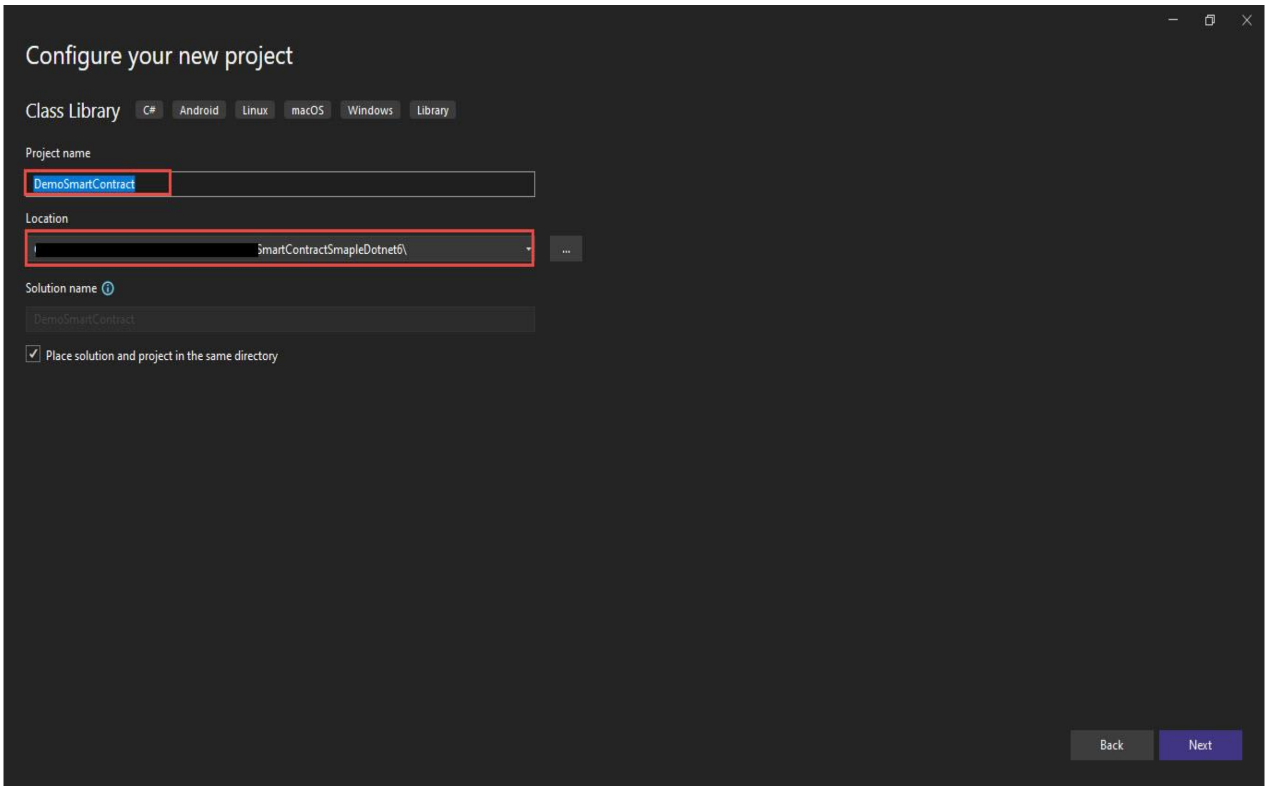
Blazor WebAssembly App
A project template for creating a Blazor app that runs on WebAssembly and is optionally hosted by an ASP.NET Core app. This template can be used for web apps with rich dynamic user interfaces (UIs).
C# Linux macOS Windows Blazor Cloud Web

ASP.NET Core gRPC Service
A project template for creating a gRPC ASP.NET Core service.
C# Linux macOS Windows Cloud Service Web

Razor Class Library

Back | Next

**Step 3**- Give the project name and directory of the project as usual.


Configure your new project

Class Library | C# | Android | Linux | macOS | Windows | Library

Project name: DemoSmartContract

Location: [redacted]SmartContractSmapleDotnet6\

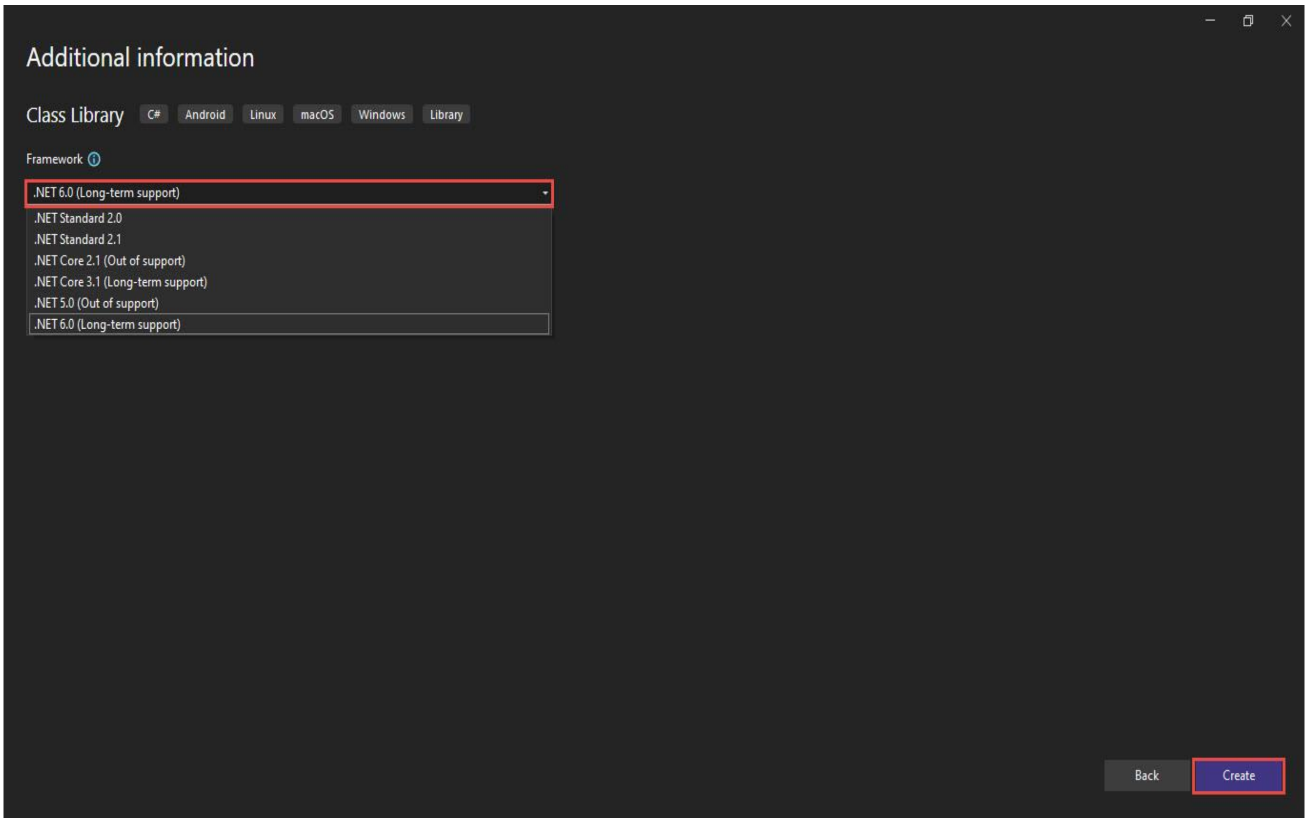
Solution name: DemoSmartContract

☑ Place solution and project in the same directory

Back | Next

**Step 4**- Select Framework **.NET 6** as shown below and Click on Create. The default empty class library project will be created as illustrated below.

Stratis


Additional information

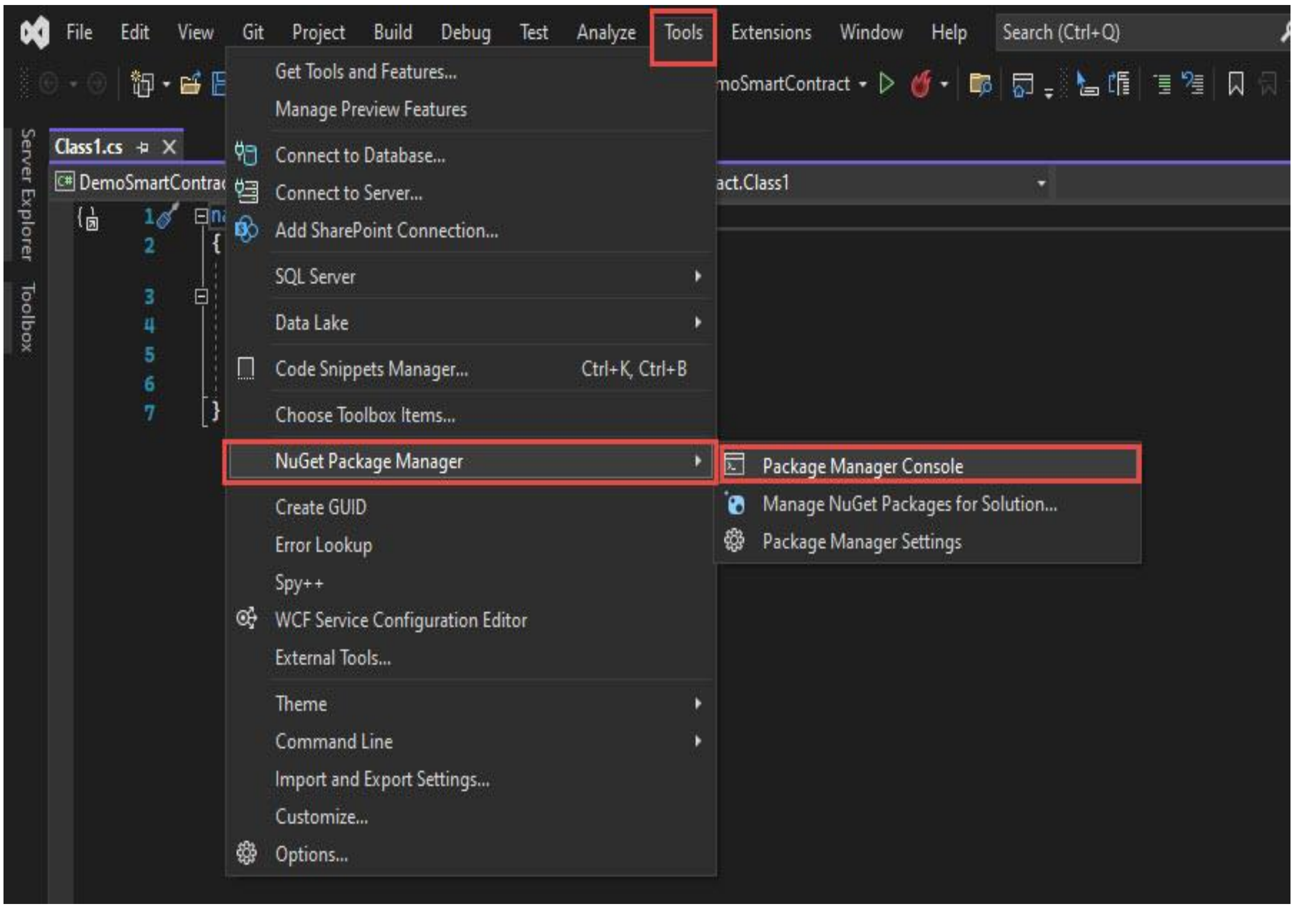
Class Library

Framework: .NET 6.0 (Long-term support)

- .NET Standard 2.0
- .NET Standard 2.1
- .NET Core 2.1 (Out of support)
- .NET Core 3.1 (Long-term support)
- .NET 5.0 (Out of support)
- .NET 6.0 (Long-term support)

**Step 5**- Now, we will install NuGet Package for Stratis Smart contract development in .NET 6. You can use the NuGet package manager console to install as usual or can use NuGet Package Manager and install from there.

Open the package manager console as shown below.

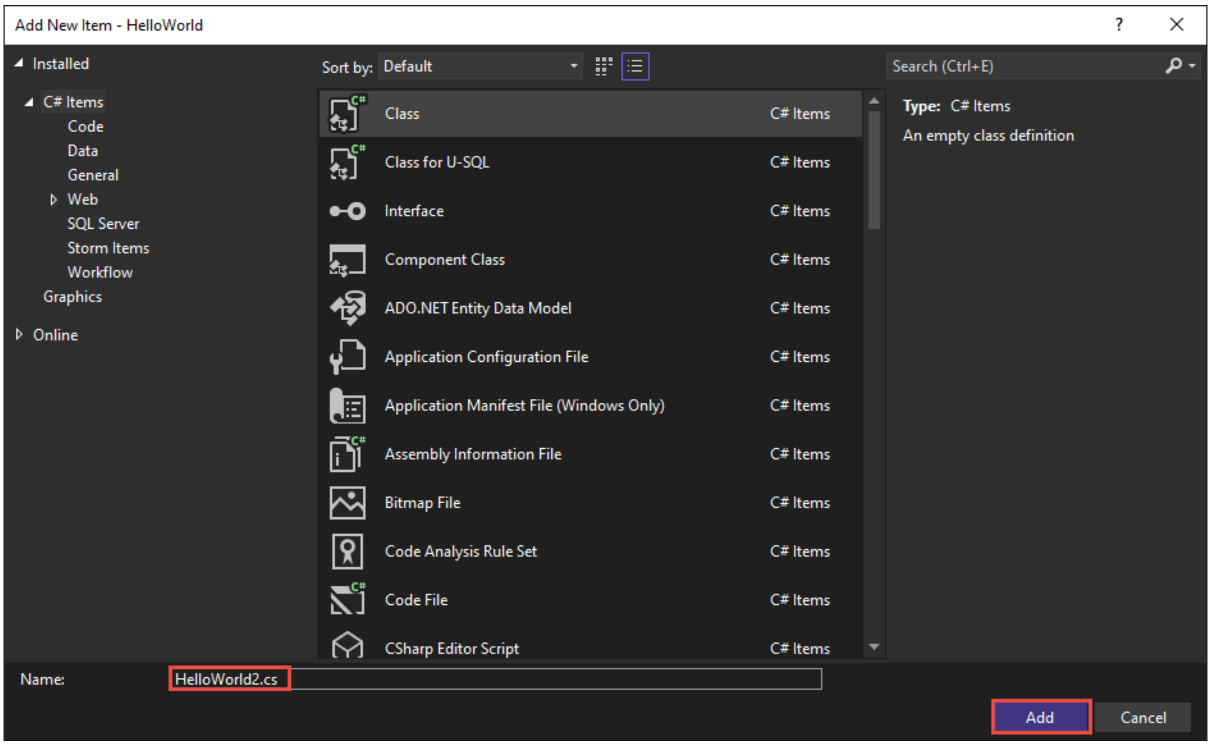


Then run the below command in the Package manager console to install Smart Contract NuGet Package for .NET 6. This package is designed for Dotnet 6 framework. You need to be careful while installing this package, so opt for the right package for .NET 6.

```
Install-Package Stratis.SmartContracts.NET6
```

## Smart Contract Program

**Step 1:** Add a new class and name it to **HelloWorld2**. Right click on project and add new class as shown below.



You can find the source code of "**Hello World 2**" for .NET 6 from the repository.

Let's begin with code.

The First line of the Contract must contain:

```
using Stratis.SmartContracts;
```

**Note:** Smart Contract class should be Public. After adding a new class, if it's not public by default, then make it public. Additionally, it shouldn't contain namespace.

All Smart contract in C# inherits from SmartContract.

```
[Deploy]
public class HelloWorld2: SmartContract
    {

    }
```

**Step 2**: Create Constructor for the class.

Let's create Constructor as like below.

```
public HelloWorld2(ISmartContractState smartContractState)
    : base(smartContractState)
    {

    }
```

Deployment of a contract involves calling the constructor of the Smart contract class.

That's why any initialization of the smart contract should be in the constructor. If we need any parameters in the Smart contract, then we can pass in the constructor.

- Smart Contract constructor gets run when the contract is created for the first time.
- Contracts must override the base class constructor and inject *ISmartContractState.*
- In constructor, the first parameters passed must be an object of the *ISmartContractState* interface.
- If you have any other parameters, then you can write after that.

**Step 3**: Add two integer properties and Changes in Greeting Property

Firstly, we need to understand the following things:

- Smart Contracts do not persist in arrays.
- However, if we want to store a group of a particular type of data we can modify and use key-value pairs to achieve this.
- Here, in the Greeting property we can use logic, but this relies on two other integer properties to maintain the array: **Index** and **Bounds**.
- We will create two integer properties: **Index** and **Bounds**

```
private int Index
    {
        get
        {
            return this.State.GetInt32("Index");
        }
        set
        {
            this.State.SetInt32("Index", value);
        }
    }

    private int Bounds
    {
        get
        {
            return this.State.GetInt32("Bounds");
        }
        set
        {
            this.State.SetInt32("Bounds", value);
        }
    }

private string Greeting
    {
        get
        {
            this.Index++;
            if (this.Index >= this.Bounds)
            {
                this.Index = 0;
            }

            return this.State.GetString("Greeting" + this.Index);
        }
        set
        {
            this.State.SetString("Greeting" + this.Bounds, value);
            this.Bounds++;
        }
    }
```

**Step 4: Add Greeting method.**

To add new greetings, we will add, AddGreeting() Method as depicted below.

```
public string AddGreeting(string helloMessage)
    {
        this.Greeting = helloMessage;
        return "Added '" + helloMessage + "' as a greeting.";
    }
```

**Step 5: Say Hello Method and Constructor**

Here, in the constructor, we will initialize the Bounds and Index along with the Greeting.

```
public HelloWorld2(ISmartContractState smartContractState)
    : base(smartContractState)
    {
        this.Bounds = 0;
        this.Index = -1;
        this.Greeting = "Hello World!";
    }
```

**SayHello()** Method will be the same as like earlier HelloWorld program which simply returns the greeting.

```
public string SayHello()
    {
        return this.Greeting;
    }
```

## Validating and Generating Bytecode

Smart contract tool (SCT) is powered by Stratis platform which is used to validate and generate the byte code of the Smart contract.

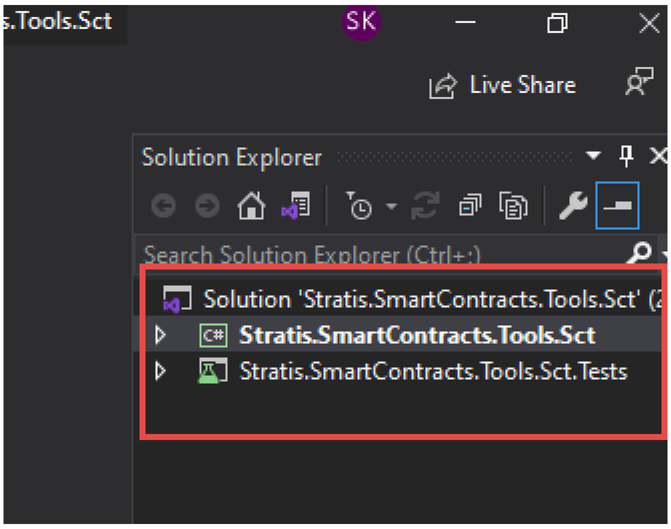
After the completion of the Smart Contract code based on the requirement, we need to validate it. Stratis has provided an Sct tool to validate the smart contract whether it is correct. The validation process is mandatory to verify the valid constraints used in the contract. It validates for the determinism and constraints used in the Smart Contract i.e. validates format and deterministic element of the contract. Determinism and format validation rules can be found in the Startis Academy. You can download the Sct tool from here.

Or clone the Stratis.SmartContracts.Tools.Sct repository by running below command.

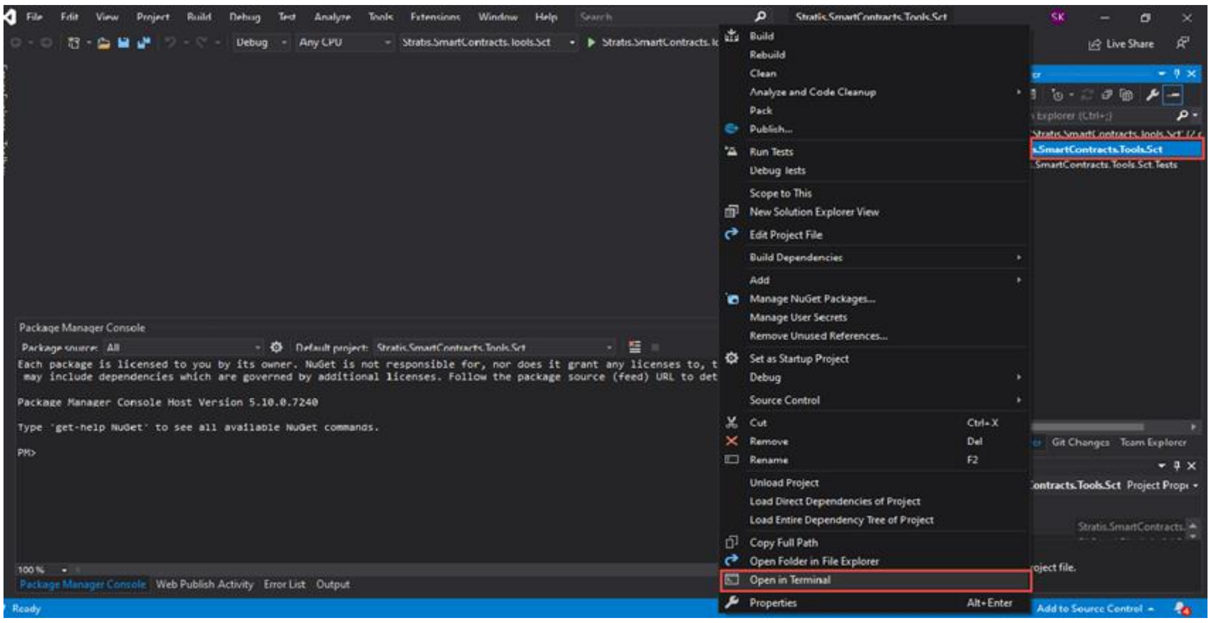
*git clone https://github.com/stratisproject/Stratis.SmartContracts.Tools.Sct.git*

**Validating Contract**

Open Sct Tools solution in Visual Studio, you will see two projects, one is the main project, and another is tests project as illustrated below.


.Tools.Sct
SK
Live Share
Solution Explorer
Search Solution Explorer (Ctrl+;)
Solution 'Stratis.SmartContracts.Tools.Sct' (2
Stratis.SmartContracts.Tools.Sct
Stratis.SmartContracts.Tools.Sct.Tests

Right-click on “Stratis.SmartContracts.Tools.Sct” and go to the terminal then run the following command.



Package Manager Console
Package source: All    Default project: Stratis.SmartContracts.Tools.Sct

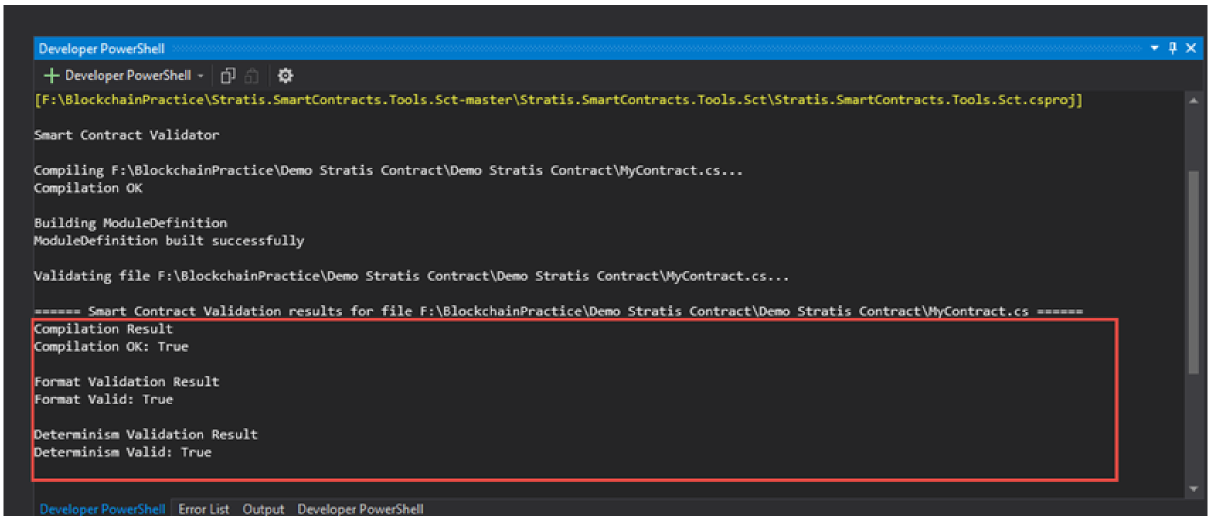
```
Each package is licensed to you by its owner. NuGet is not responsible for, nor does it grant any licenses to, t
 may include dependencies which are governed by additional licenses. Follow the package source (feed) URL to det

Package Manager Console Host Version 5.10.0.7240

Type 'get-help NuGet' to see all available NuGet commands.

PM>
```

Let’s validate the contract running the below command in the terminal. Note that the validating and compiling step is same for all contracts.

*dotnet run -- validate "E:\Extras\Projects forBook\HelloWorld\HelloWorld\HelloWorld2.cs"*

Once you run the command you can see the success or error of validation. On the success of validation, you will get the below information in the terminal.

Stratis



```
[F:\BlockchainPractice\Stratis.SmartContracts.Tools.Sct-master\Stratis.SmartContracts.Tools.Sct\Stratis.SmartContracts.Tools.Sct.csproj]

Smart Contract Validator

Compiling F:\BlockchainPractice\Demo Stratis Contract\Demo Stratis Contract\MyContract.cs...
Compilation OK

Building ModuleDefinition
ModuleDefinition built successfully

Validating file F:\BlockchainPractice\Demo Stratis Contract\Demo Stratis Contract\MyContract.cs...

====== Smart Contract Validation results for file F:\BlockchainPractice\Demo Stratis Contract\Demo Stratis Contract\MyContract.cs ======
Compilation Result
Compilation OK: True

Format Validation Result
Format Valid: True

Determinism Validation Result
Determinism Valid: True
```

If you get an error in validation, then based on the error message you can correct your smart contract code and then again do validation as above.

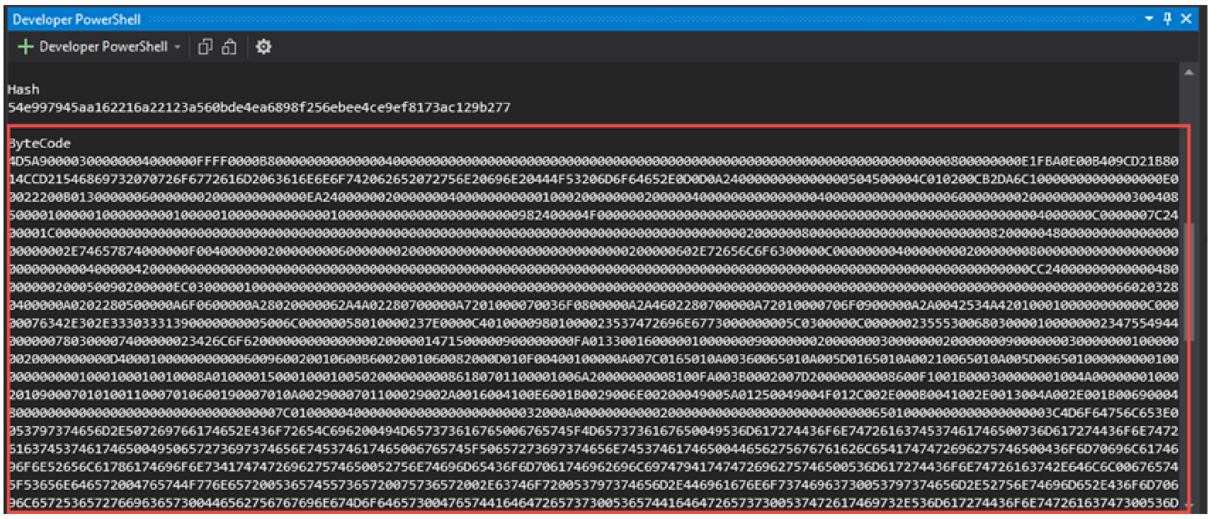
## Compiling and Generating Byte Code

Once, Contract is validated, after that, we must compile the Smart Contract code and generate the byte code. This smart contract byte code is the code that we need to deploy in the Blockchain.

Now, Generate Byte Code running the example command given below.

*dotnet run -- validate "E:\Extras\Projects forBook\HelloWorld\HelloWorld\HelloWorld2.cs" -sb*

The above command first validates the Smart contract and then compiles the code. On the success of your compilation, you will get hash and byte code in your terminal as illustrated below. We will need the hash and byte code while deploying the contract in the blockchain. So, copy and keep the hash and Contract Byte code which we will use in next part of the hands-on.



```
Hash
54e997945aa162216a22123a560bde4ea6898f256ebee4ce9ef8173ac129b277

ByteCode
4D5A90000300000004000000FFFF0000B800000000000000400000000000000000000000000000000000000000000000000000000000000000000000800000000E1FBA0E00B409CD21B80
14CCD21546869732070726F6772616D2063616E6E6F742062652072756E20696E20444F53206D6F64652E0D0D0A2400000000000000504500004C010200CB2DA6C1000000000000000E0
0022200B013000000600000002000000000000EA240000002000000040000000000100020000000200000400000000000000040000000000000006000000000200000000000000300408
500001000000100000000010000001000000000000001000000000000000000000982400004F000000000000000000000000000000000000000000000000000004000000C0000007C24
00001C0000000000000000000000000000000000000000000000000000000000000000000000000000000000200000080000000000000000000000000000082000004800000000000000
00000002E746578740000000F00400000002000000006000000020000000000000000000000000000200000602E72656C6F6300000C00000000400000000200000008000000000000000
000000000040000004200000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000CC240000000000000480
000000200050090200000EC03000001000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000006602032B
0400000A0202280500000A6F0600000A28020000062A4A022807000000A72010000070036F0800000A2A460228070000000A72010000706F0900000A2A0042534A4201000100000000000C000
00076342E302E33303331390000000005006C00000058010000237E0000C40100000980100002353747269E67730000000005C0300000C000000023555300680300001000000002347554944
0000000780300000740000000023426C6F620000000000000000000200000147150000000900000000FA0133001600000010000000090000000200000000300000000200000000900000000300000000100000
0020000000000D40001000000000000600960020010600860020010600820000D010F00400100000A007C0165010A00360065010A005D0165010A00210065010A005D00650100000000001000
000000000100010001001008A010000150001000100502000000000861807011000010006A20000000008100FA003B0002007D200000000008600F1001B000300000001004A00000001000
2010900070101001100070106001900070100A00290007011002900200A0016004100E6001B0029006E00200049005A01250049004F012C002E000B0041002E0013004A002E001B00690004
000000000000000000000000000000000007C0100000400000000000000000000000032000A00000000000002000000000000000000000000650100000000000000000003C4D6F64756C653E0
0537973746564D2E507269766174652E436F72654C696200494D65737361676500676574F4D657373616765004953616D617274436F6E7472616374537461746500736D617274436F6E7472
5163745374617465004950657273697374656E745374617465007065F506572736973746E74537461746500446562756767610626C65417474726962757465004436F6D70696C61746
96F6E52656C61786174696F6E734174747269627574650052756E74696D65436F6D7061746962696C697479417474726962757465005365744D657373616765746F4D657373616765006D
5F53656E646572004765744F776E6572005365745573657200757365720002E63746F72005379737465D2E446961676E6F737469637300537973746D2E52756E74696D652E436F6D70
6C657253657276696365730044656275676796E674D6F6465730047657441646472657373005365744164647265737300537472617469732E536D617274436F6E74726163747300536D
```

**Note**: If you are going to deploy the contract on **test net** or **main net** then the hash of the Contract is needed to be whitelisted by master node operator. If you are going to deploy in your private net instance, then no need to whitelist it.

# Hands-on 3: Deploying and Interacting with Extended Hello World Contract on Stratis Blockchain with Cirrus Core Wallet

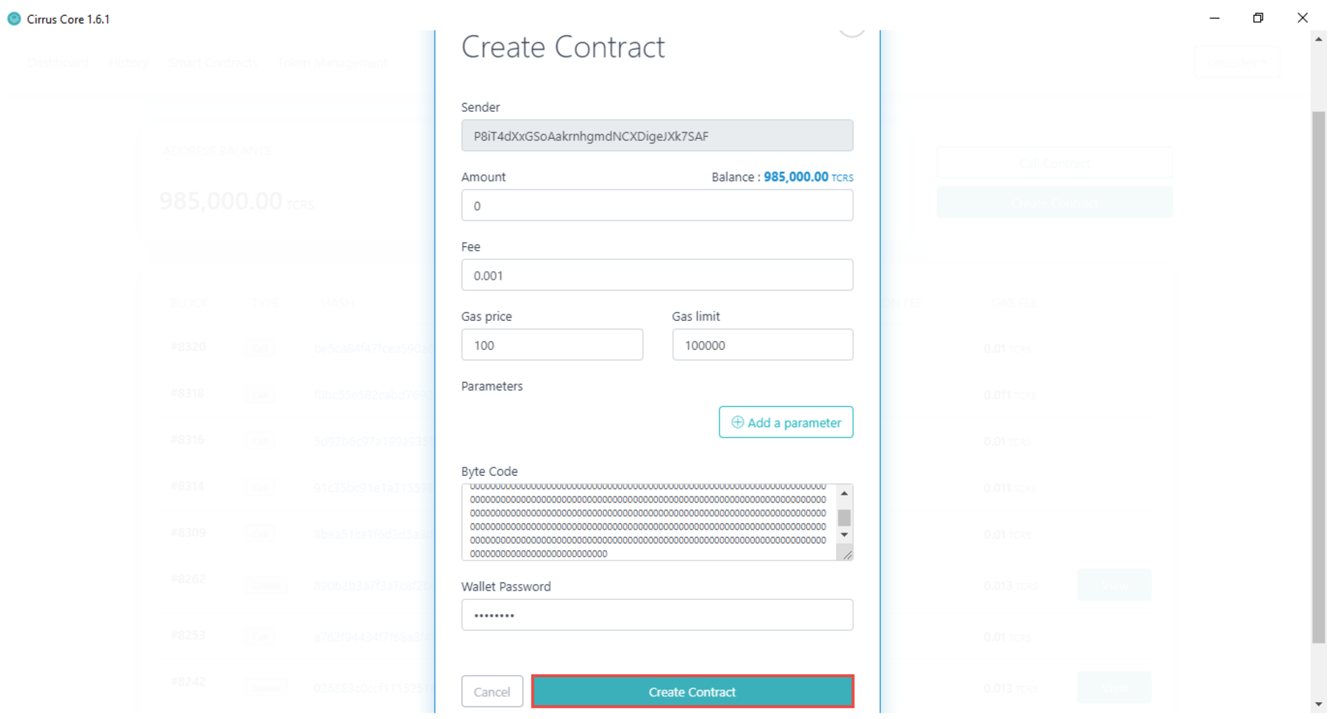
In this section we will deploy the Contract on the Blockchain. We will perform following:

- Deploy and interact with the Contract.
- Add multiple greetings.
- Call a method on the contract.

**Step 1: Deploy the Contract**

To deploy a contract on Stratis Blockchain, open wallet and go to Smart Contract Tab. After that click on Create Contract.

Then, provide the Contract Bytecode, password of the wallet, and parameters if needed, and click on Create Contract as in earlier deployment.



Sender: P8iT4dXxGSoAakrnhgmdNCXDigeJXk7SAF

Amount: 0 — Balance : 985,000.00 TCRS

Fee: 0.001

Gas price: 100

Gas limit: 100000

Parameters

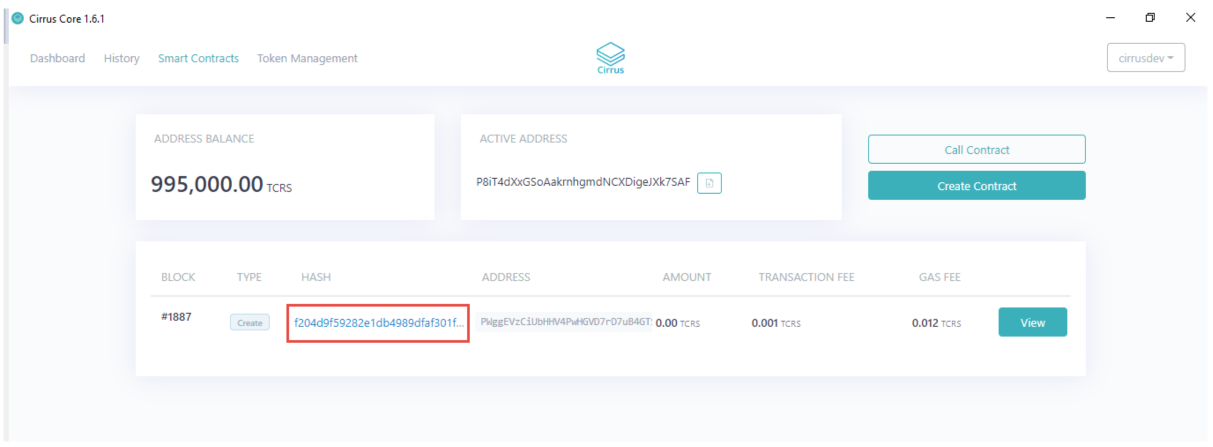
Byte Code: 0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000

Wallet Password: ••••••••

Once, deployment is successful, you can get the receipt of the transaction where you can see the new Contract Address. The new Contract Address is your deployed contract address. Alternatively, you can see the deployed Extended Hello world contract address on the Smart Contract Dashboard as well.

**Step 2: Check Receipt of Contract Deployment**

You can view the receipt of your transaction by clicking on hash as depicted below.



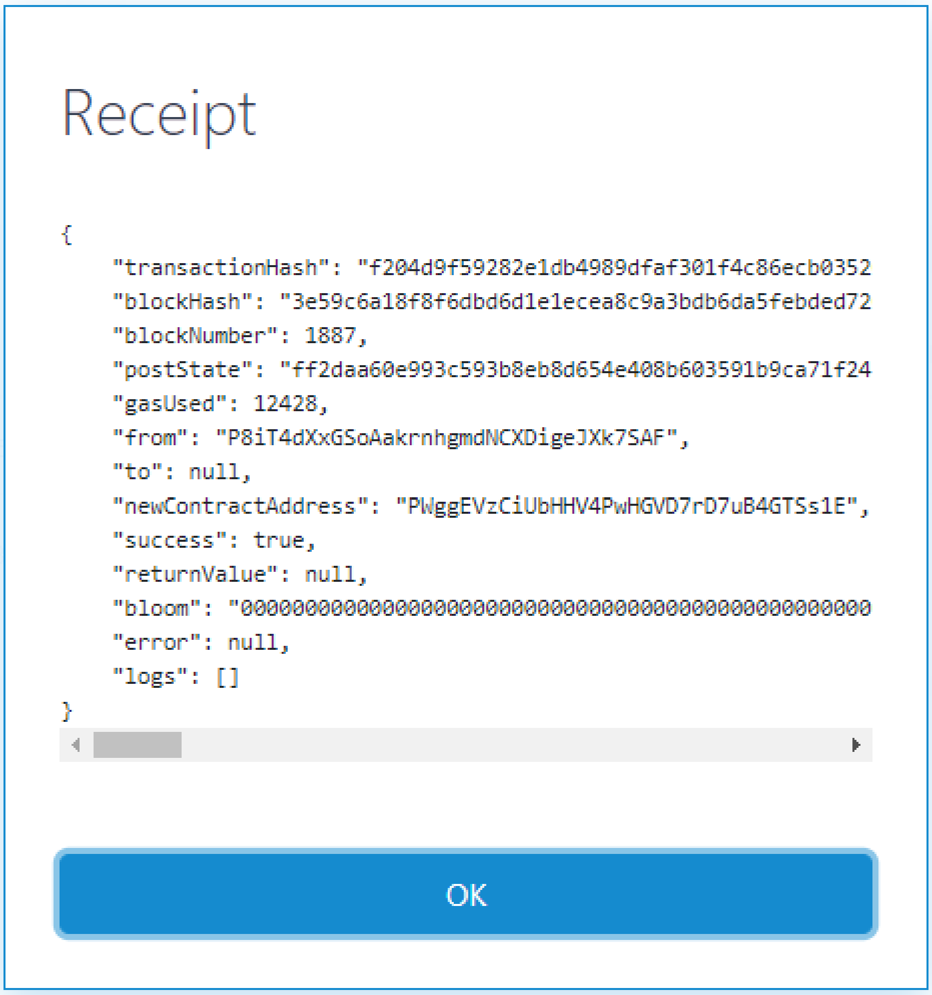
ADDRESS BALANCE

995,000.00 TCRS

ACTIVE ADDRESS

P8iT4dXxGSoAakrnhgmdNCXDigeJXk7SAF

| BLOCK | TYPE | HASH | ADDRESS | AMOUNT | TRANSACTION FEE | GAS FEE | |
|---|---|---|---|---|---|---|---|
| #1887 | Create | f204d9f59282e1db4989dfaf301f... | PWggEVzCiUbHHV4PwHGVD7rD7uB4GT: | 0.00 TCRS | 0.001 TCRS | 0.012 TCRS | View |



```
{
    "transactionHash": "f204d9f59282e1db4989dfaf301f4c86ecb0352
    "blockHash": "3e59c6a18f8f6dbd6d1e1ecea8c9a3bdb6da5febded72
    "blockNumber": 1887,
    "postState": "ff2daa60e993c593b8eb8d654e408b603591b9ca71f24
    "gasUsed": 12428,
    "from": "P8iT4dXxGSoAakrnhgmdNCXDigeJXk7SAF",
    "to": null,
    "newContractAddress": "PWggEVzCiUbHHV4PwHGVD7rD7uB4GTSs1E",
    "success": true,
    "returnValue": null,
    "bloom": "000000000000000000000000000000000000000000000000
    "error": null,
    "logs": []
}
```

**Note**: if you face any issue deploying and calling contract, you can see the errors in the receipt itself.

Furthermore, once the Cirrus Core Private Net Wallet is running in your machine, you can interact and explore with the RESTFul API through Swagger available on [http://localhost:38223/Swagger](http://localhost:38223/Swagger).

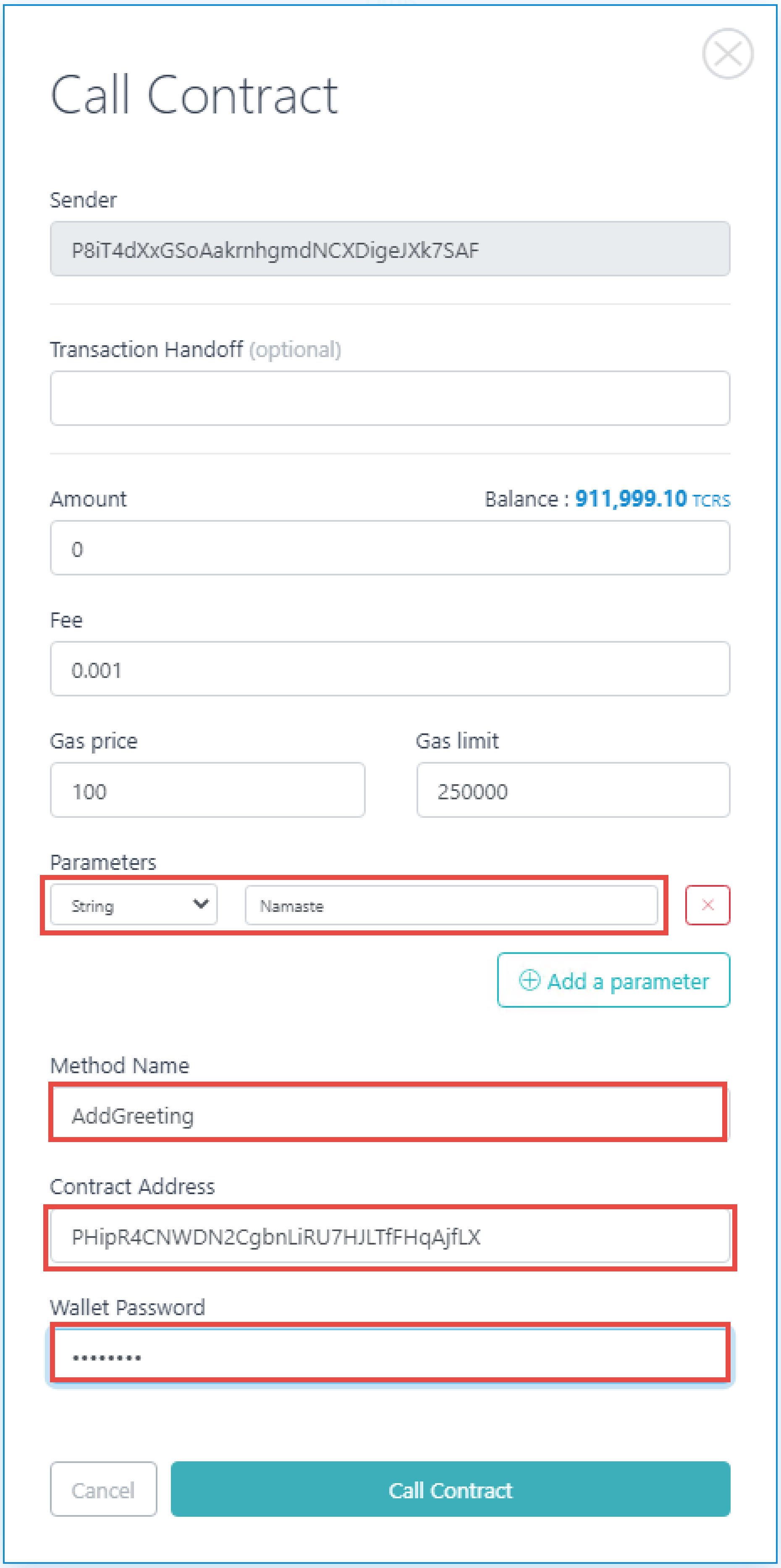
**Step 3: Call Contract Method- AddGreeting()**

Now, we will add multiple greetings calling the contract method.

Click on the Call Contract button, then provide Method Name: **AddGreeting**, Contract Address, Wallet password, and add below Greeting in different languages as a parameter.

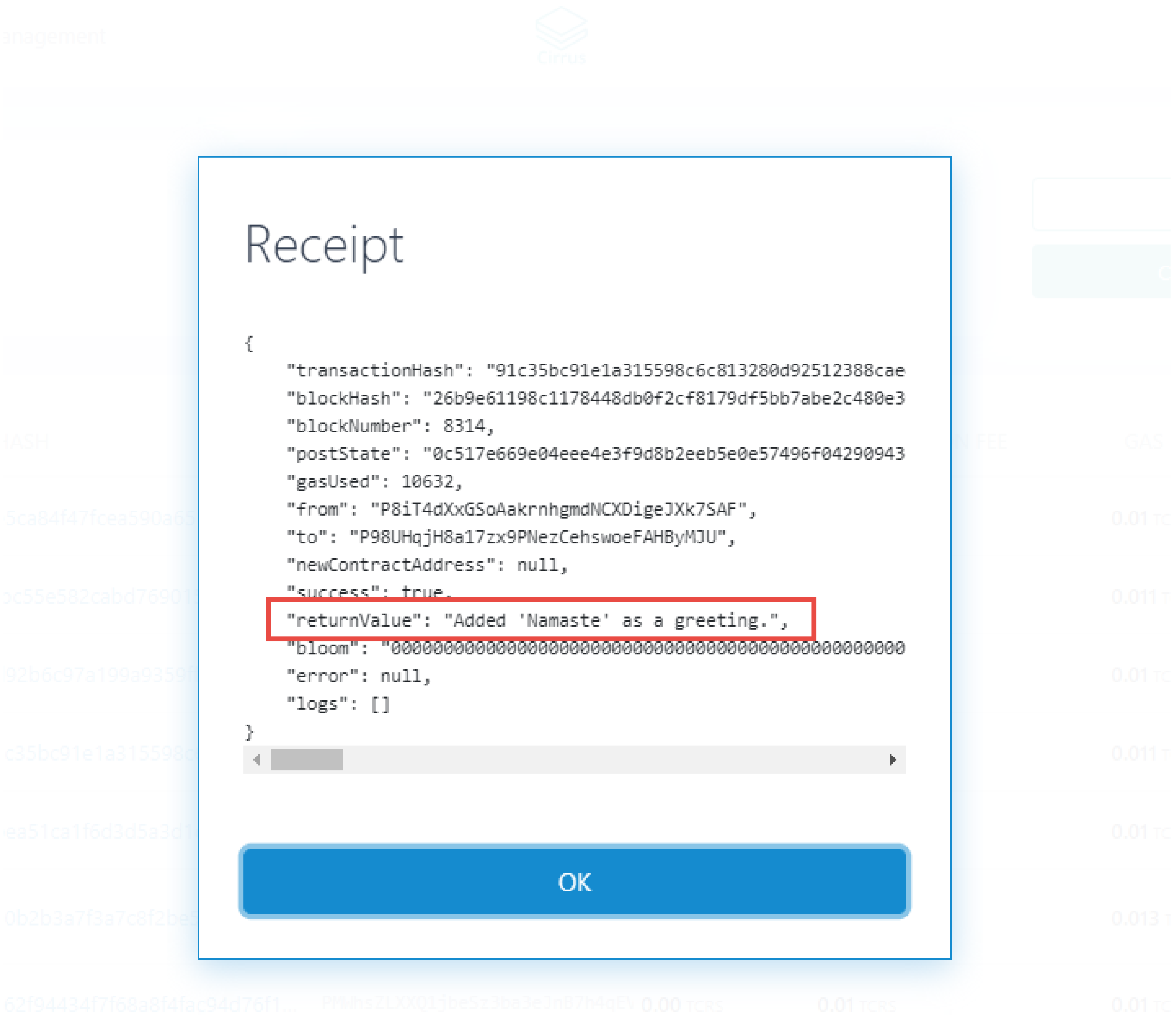
- **Namaste! -  Indian/Nepali**
- **Hallo Welt! - German**
- **Привет, мир! - Russian**
- **你好，世界！ - Chinese**
- **Hej Verden! – Danish**

Add one greeting value at a time see the below example of adding a Greeting: **Namaste!**


## Call Contract

| | |
|---|---|
| Sender | P8iT4dXxGSoAakrnhgmdNCXDigeJXk7SAF |
| Transaction Handoff (optional) | |
| Amount | 0 |
| Balance : | 911,999.10 TCRS |
| Fee | 0.001 |
| Gas price | 100 |
| Gas limit | 250000 |
| Parameters | String — Namaste |
| Method Name | AddGreeting |
| Contract Address | PHipR4CNWDN2CgbnLiRU7HJLTfFHqAjfLX |
| Wallet Password | •••••••• |

Once the greeting is added, you can see the receipt by clicking on Hash and it will be like below.



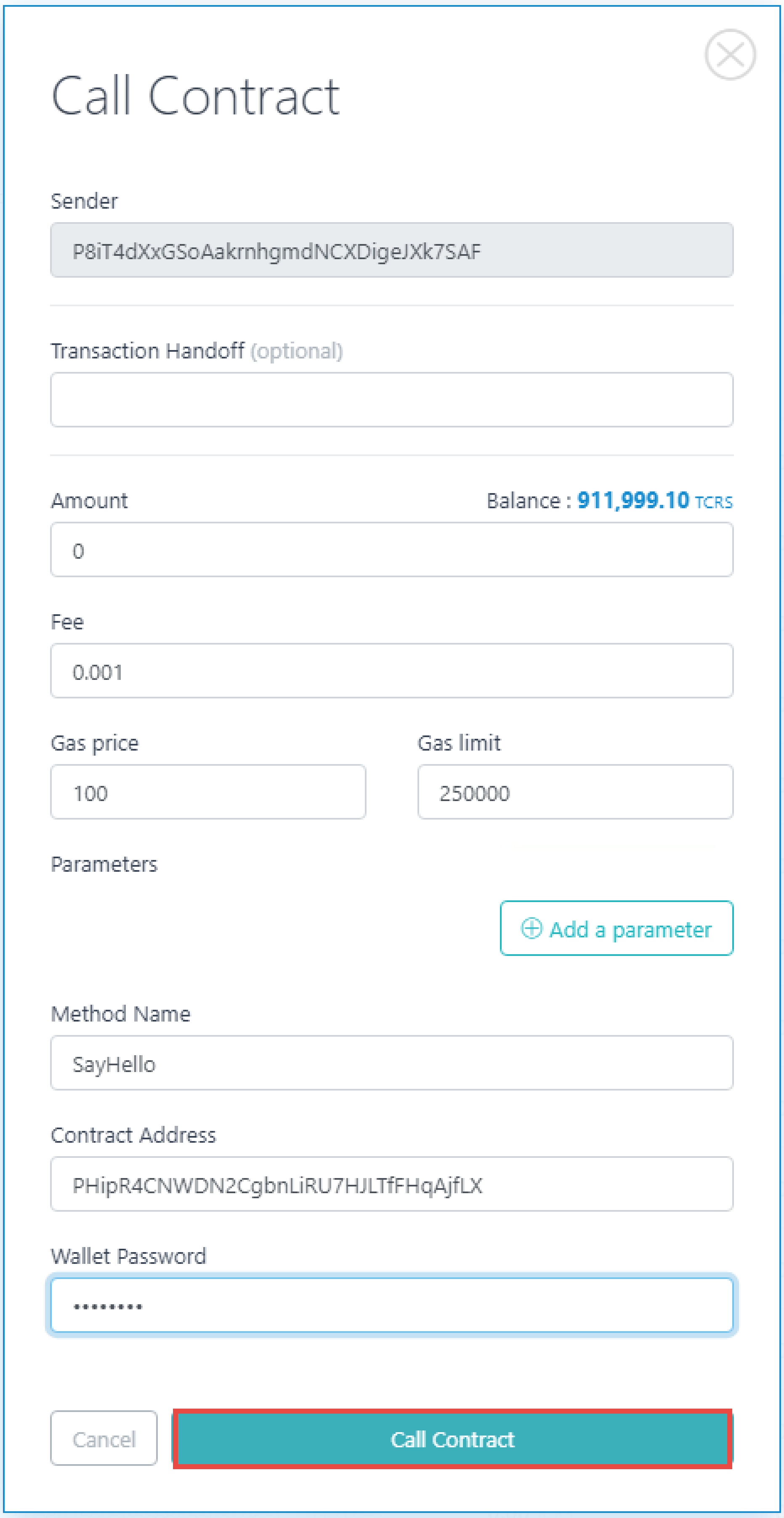
Receipt

```
{
    "transactionHash": "91c35bc91e1a315598c6c813280d92512388cae
    "blockHash": "26b9e61198c1178448db0f2cf8179df5bb7abe2c480e3
    "blockNumber": 8314,
    "postState": "0c517e669e04eee4e3f9d8b2eeb5e0e57496f04290943
    "gasUsed": 10632,
    "from": "P8iT4dXxGSoAakrnhgmdNCXDigeJXk7SAF",
    "to": "P98UHqjH8a17zx9PNezCehswoeFAHByMJU",
    "newContractAddress": null,
    "success": true,
    "returnValue": "Added 'Namaste' as a greeting.",
    "bloom": "00000000000000000000000000000000000000000000000000
    "error": null,
    "logs": []
}
```

Repeat this step and add multiple greetings.

**Step 4: Call Contract Method SayHello()**

Now, let’s call the SayHello method. For this Click on the Call Contract button, then provide the Method Name: SayHello, Contract Address, Wallet password, and click on Call contract.



Call Contract

Sender

P8iT4dXxGSoAakrnhgmdNCXDigeJXk7SAF

Transaction Handoff (optional)

Amount Balance : 911,999.10 TCRS

0

Fee

0.001

Gas price

100

Gas limit

250000

Parameters

Method Name

SayHello

Contract Address

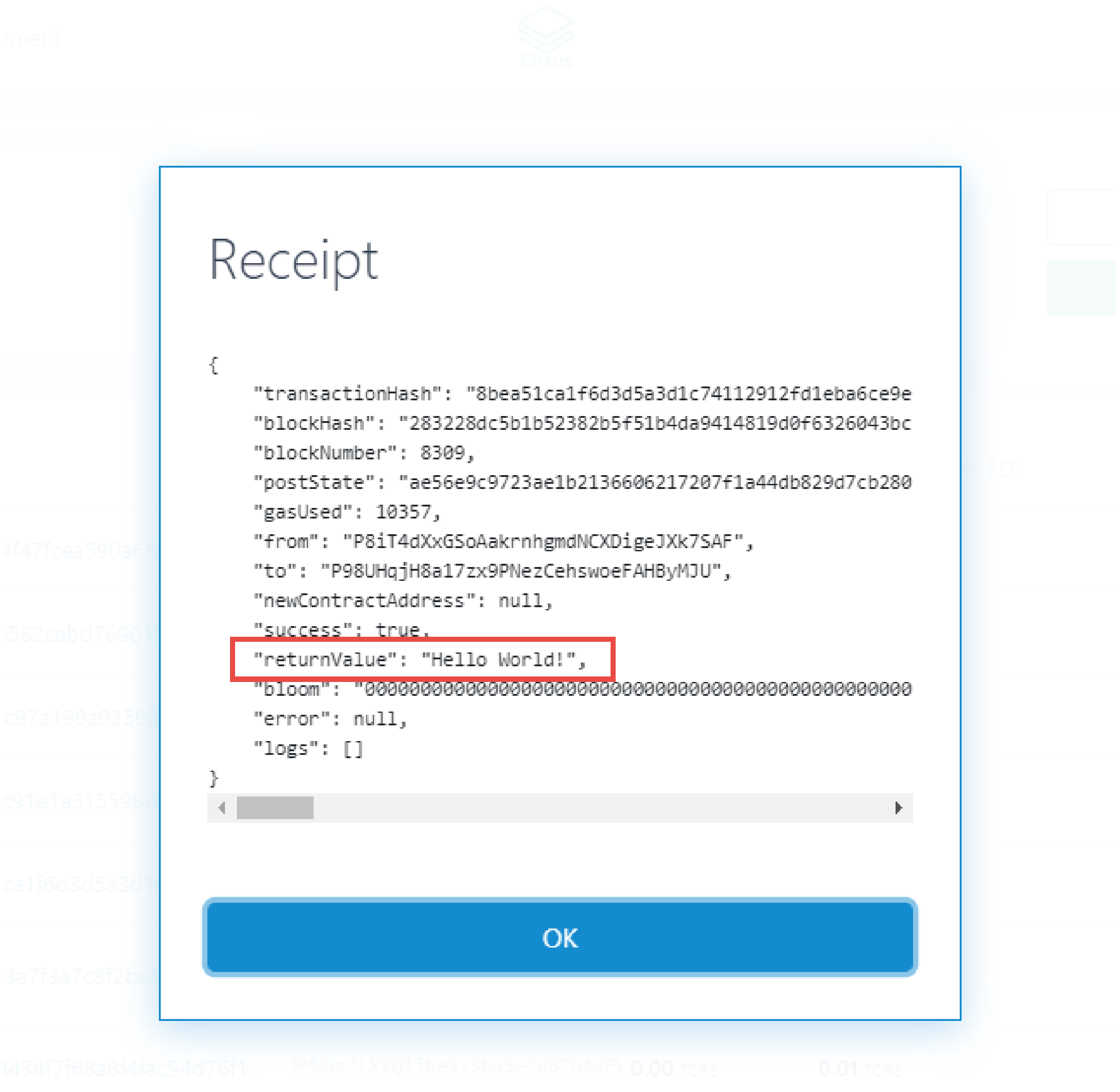
PHipR4CNWDN2CgbnLiRU7HJLTfFHqAjfLX

Wallet Password

•••••••••

It will cycle through the greetings.

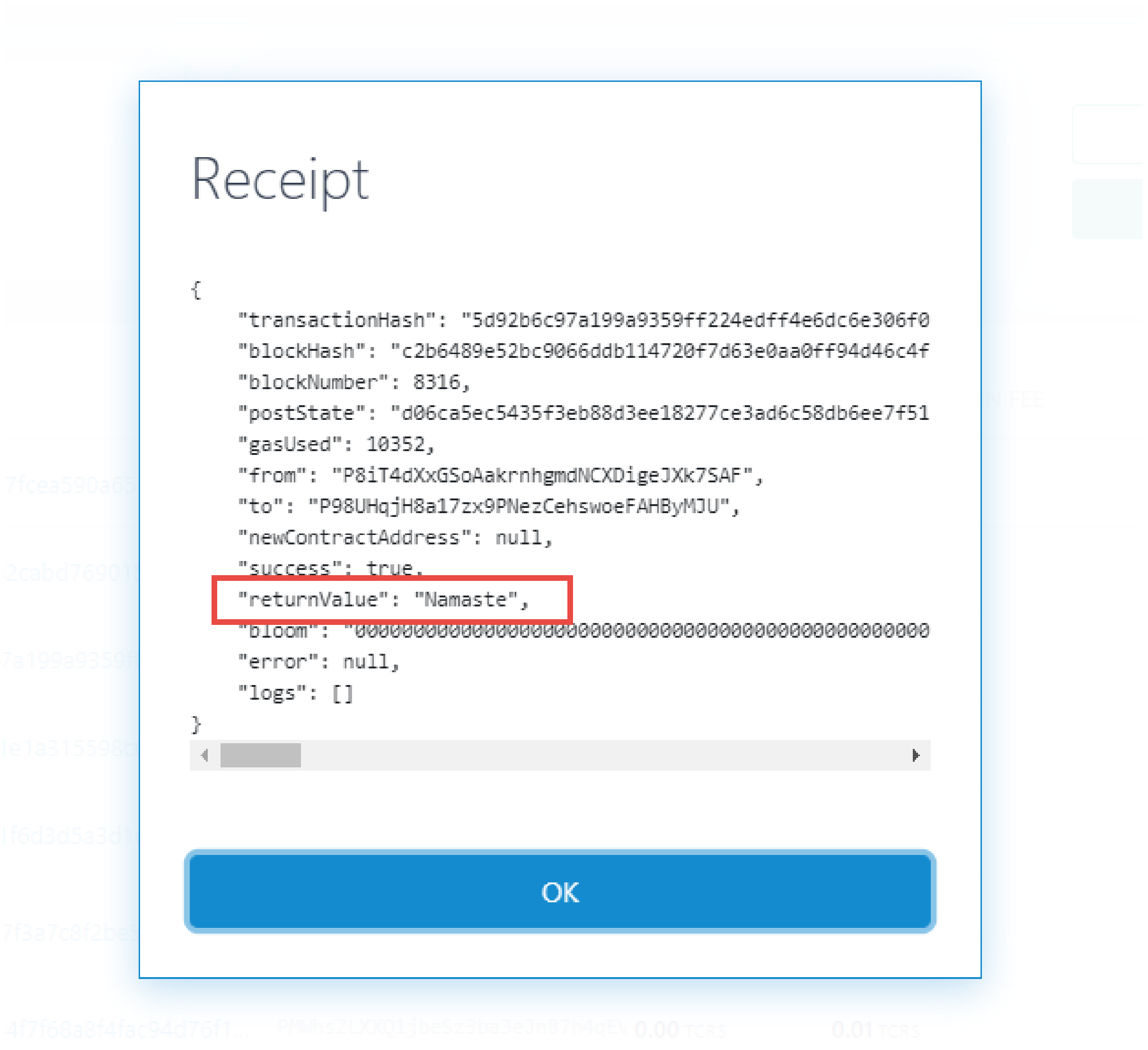
When you call the SayHello method for the first time, it will display: **“Hello World!”**



Receipt

```
{
    "transactionHash": "8bea51ca1f6d3d5a3d1c74112912fd1eba6ce9e
    "blockHash": "283228dc5b1b52382b5f51b4da9414819d0f6326043bc
    "blockNumber": 8309,
    "postState": "ae56e9c9723ae1b2136606217207f1a44db829d7cb280
    "gasUsed": 10357,
    "from": "P8iT4dXxGSoAakrnhgmdNCXDigeJXk7SAF",
    "to": "P98UHqjH8a17zx9PNezCehswoeFAHByMJU",
    "newContractAddress": null,
    "success": true,
    "returnValue": "Hello World!",
    "bloom": "000000000000000000000000000000000000000000000000
    "error": null,
    "logs": []
}
```

Then, in the second-time call, it will display: “**Namaste”** and so on based on the Added greeting messages, and it cycles through.



Receipt

```
{
    "transactionHash": "5d92b6c97a199a9359ff224edff4e6dc6e306f0
    "blockHash": "c2b6489e52bc9066ddb114720f7d63e0aa0ff94d46c4f
    "blockNumber": 8316,
    "postState": "d06ca5ec5435f3eb88d3ee18277ce3ad6c58db6ee7f51
    "gasUsed": 10352,
    "from": "P8iT4dXxGSoAakrnhgmdNCXDigeJXk7SAF",
    "to": "P98UHqjH8a17zx9PNezCehswoeFAHByMJU",
    "newContractAddress": null,
    "success": true,
    "returnValue": "Namaste",
    "bloom": "000000000000000000000000000000000000000000000000000
    "error": null,
    "logs": []
}
```

Hence, in this hands-on, we got an overview of the Extended Hello world contract, deployed it, and called the Contract method.

## Reference

1. https://academy.stratisplatform.com/Developer%20Resources/SmartContracts/Tutorial3-ExtendingHelloWorld/hello-world-extension-tutorial-introduction.html