

Simple Smart Contract Install and Setup

This document outlines the steps needed to develop, test, launch and configure this generic Selene Network compatible smart contract.

Prerequisites for Building

You can launch this contract without making changes, or you can make changes.

Unaltered

If you simply want to launch a default, unaltered, version of this contract, you'll need to have the willingness to use ThetaScan's "Deploy Contract" and "Interact With" functionality within it's "Smart Contract HQ".

The bytecode and ABI are provided within this zip file in the project directory. Instructions for configuring the contract once it is launched is provided below.

Altered

If you want to make modifications to this generic smart contract, you'll need to have a basic understanding of Solidity coding along with the ability to use Remix provided by ThetaScan.io. That "Remix for Theta" can be found in the "Smart Contract HQ."

Information about how to test your modifications is outlined below in the Building and Testing section.

Feature Set

This sample is built off the Open Zeppelin ERC721 & ERC721Metadata examples. It has all the basic functionality that you would expect in a standard NFT and more.

In order to integrate into the Selene Network, it also supports a few other interfaces that allow for the common experience that you see when you visit projects that are displayed by the Selene Network code.

Here are some of the extra details provided in this example code.

- Minting a limited or unlimited supply
- Ability to assign per-token based metadata
- Ability to toggle minting on or off
- Can be soulbound or fully transferable
- Built in sendLock functionality (safety feature)
- Burnable

There is additional functionality that can be referenced after launching.

- Adjust overall price (in pennies) after launch
- Relocate project after launch
- Project owner can mint without commissions

Launching

The easiest way to get up and running is to use the bytecode and ABI that is already provided and simply deploy the contract using ThetaScan.

Because it costs a few Tfuel to launch a contract, you'll need to connect your Metamask wallet to the ThetaScan website. Make sure you set the active account to the account that you want to be the owner of the contract.

To launch the contract, visit ThetaScan's 'Smart Contract HQ' and click on 'Deploy Contract.' When that page displays, it will present two edit controls: one for the ABI and the other for the bytecode.

The data for both of these edit controls can be found within this project zip file. They are named *drmaudiov1_abi.json* and *drmaudiov1_bytecode.txt*. Just cut and paste the data from these files into the appropriate edit control.

Once done, press next and you'll be presented with the collection of parameters that are required by the contract. It may seem large, but we'll cover each one here.

Deploy a Contract

Constructor Variables :

<input type="text" value="_theTitled"/>
<input type="text" value="_thePennyOracle"/>
<input type="text" value="_theParticipant"/>
<input type="text" value="_theAgent"/>
<input type="text" value="_theWebsite"/>
<input type="text" value="_strProjectName"/>
<input type="text" value="_bSoulbound"/>
<input type="text" value="_bSendLock"/>
<input type="text" value="_limit"/>

Estimated Transaction Fee:

Estimated Value in USD: \$

Transaction Hash:

_theTitled

The Selene Network compatible smart contracts allow for two types of owners. One has access to the balance on the contract and the other has access to managing the functionality of the contract. The relationship is like that of a boss and a worker. The

worker can make changes, but the boss collects the money. The boss is `_theTitled` and the worker is the account that launches the contract.

At this point, double check that the account that is currently active in Metamask. Whatever the active account is will be the owner of the contract.

Supply as `_theTitled` parameter a Metamask Theta wallet address where you want to collect your funds.

Note that it is ok to have the same address for both the contract owner and `_theTitled`.

`_thePennyOracle`

The best way to get the Penny Oracle contract address is to use the tools functionality that is built into the Selene Network core code. The ‘hosting’ document talks about this in detail. When processing your project JSON file, that code will display the contract addresses for the Penny Oracle, the Participant, the Agent and Website contracts.

Or, you can fetch the addresses off any Selene Network rendered page. The information is located in the footer.

For the Penny Oracle address, look for the hyperlink to “Penny Oracle” when clicked, that link will take you to the Theta explorer showing you the Penny Oracle account.

Select that contract address for `_thePennyOracle` and place it in this field.

`_theParticipant`

For `_theParticipant`, `_theAgent` and `_theWebsite` contract addresses, see the links in the footer like how you got the Penny Oracle contract address. All three of these parameters are simply the contract addresses of each respective contract.

Or, simply view your project JSON file using the tools functionality and the contract address will be displayed.

`_theAgent`

See `_theParticipant`.

`_theWebsite`

See `_the Participant`.

`_strProjectName`

The name that you provide here is a standard short string of characters. In the case of this sample project “drmaudiov1”.

Note that the name provided here must match the project JSON file that you author that goes along with the project. It will also be the subdirectory name on the server that hosts the metadata files.

Thus, expect to host your project somewhere like:

`https://YourServer.ext/nfts/album_moon` if your project were called “album_moon”.

Tip

Please read the document regarding “How to Host a Project” which is included in this ZIP file.

`_bSoulbound`

This can be either true or false. If set true, any token that is minted will not be transferable. It will only live in the account that mints the token. An example of this functionality is the Participant NFT that is used by the Selene Network to tokenize the high level Selene Network code.

Generally, `_bSoulbound` would be false.

`_bSendLock`

Every transferable Selene Network compatible smart contract offers the ability to preschedule a send transaction which can be used as insurance for ownership. For non-soulbound tokens, when this is set true the high level code will offer the owner the ability to register a backup address for the token. This is done to help protect ownership of valuable assets that must be held in a hot wallet like Metamask. Once the owner registers a `sendLock` backup address, the next time a send operation is performed on the token, it will move to the pre-registered wallet address.

The `_bSendLock` functionality is best suited for smart contracts that need to hold assets in hot wallets in order to unlock functionality for the user.

Non-transferable tokens should have `_bSendLock` set to false, for it can not be used.

`_limit`

This field is an integer number. If it is set to zero (0) there will be no limit to the number of Tokens that can be minted. If you need to limit the count, provide a natural number that represents the maximum amount.

Even though an upper limit can be set during the configuration of the contract, the project owner has the ability to mint beyond this limit.

Estimating Gas

Once you have filled in the required parameters, you can click the ‘Estimated Gas’ button to determine the cost of launching the contract.

Deploy

When you’re ready, press the ‘Deploy’ button and ThetaScan will launch Metamask to ask your permission to launch the contract. Once you confirm that operation, the information will be sent to the blockchain for submission. After about 15-20 seconds, you will be provided the contract address.

Save that contract address. This is the address of your Selene Network compatible smart contract.

Post Launch Configuration

After launching the contract, you need to prepare the hosting so that when you configure the smart contract the data points to something useful. If you haven’t already read the accompanying document “How to Host a Project”, you should read that now.

It is assumed that the reader has finished reading the reference document and they are ready to enable the contract to reference the hosting location.

Preparing the Smart Contract

After launching the contract and having prepared the server, there are two functions that you need to call in order to provide information used for minting and setting the project metadata. No minting can occur until both of these functions have been called.

`payInitialize()`

This function is used to configure the cost and commissions for the contract. Note that it can only be called once, thus get the parameters correct the first time. All parameters are natural numbers.

`mintPennies`

This is the number of pennies (in US dollars) that are required to mint the token. Selene Network projects use the Penny Oracle to get the current cost of a penny's worth of tfuel in order to dynamically update the amount of coin required for minting based on the US dollar price. If you want the Token mintable for \$5, you'd put 500 as the mintPennies parameter.

`timeoutDays`

This parameter is used for paying commissions. The agent will have this number of days in order to satisfy the minCount needed to withdraw funds. After this time, the project owner can 'sweep' the account to claim unrealized commissions left behind by agents.

Even if you have a time sensitive mint, or the project owner turns off minting, it generally makes sense to make this number fairly large. Making it large indicates to the Agent selling the tokens that you will give them a long time to claim any commissions.

Making this value a year plus a day or even two years makes sense.

`minCount`

This is the number of times that an Agent or Website address needs to be involved in minting transactions in order to qualify for the commissions. The idea aligns to giving a discount for making multiple sales rather than individual sales.

It's reasonable to make this count 1, 2 or even 3.

`cutPercentAgent`

When offering commissions to agents, it is done through this parameter. This field is expected to be a value from 0 to 100. The calculation is based off the mintPennies value above.

If a project creator needs to pocket 65 cents on the dollar, they might offer 30% commission for agents and 5% for websites. In which case, the number for cutPercentAgent would be 30.

Note that the value carved out for agents is split between the founding agent and the partner agent in a 10/90 ratio.

cutPercentWebsite

When offering commissions to websites that support hosting and access to the Selene Network, it's done through this cut. Like with cutPercentAgent, the cutPercentWebsite is the amount of the mintPennies that should be cut out for the website commission.

In the example above that pays 5% for websites, the project creator would be 5 in this field.

Ultimately, when the code carves out the splits, it carves out the agent percent, then the website percent and records whatever is left over to the project creator.

projectUpdate()

This function requires data that defines where the project metadata can be found. It can be called multiple times and can be used to update the project if the project moves hosting locations. It can also be called after changes are made to project files that require recording a new project hash.

_URI

This parameter is the location where the project file metadata can be found.

An example might be: <https://amorstyle.com/nfts/drmaudiov1/>

Note that there is a trailing slash.

_Project

This is the name of the project JSON file, like 'drmaudiov1' in the example used on this document.

This name in combination with the _URI like:

<https://amorstyle.com/nfts/drmaudiov1/drmaudiov1.json>

should resolve. That is the location and file that is read to display information about the project.

_hash

The value that needs to be provided here is the SHA1 hash of the project JSON file. In this example, download the file: <https://amorstyle.com/nfts/drmaudiov1/drmaudiov1.json> to your windows machine and use the certification tool to hash it.

```
certutil -hashfile drmaudiov1.json SHA1
```

Using tools to get projectUpdate() info

The Selene Network provides a tools page that can be used to process the project JSON file to get the above information. An example of how to do this is like this:

<https://amorstyle.com/dsn/tools/?project=https://amorstyle.com/nfts/drmaudiov1/drmaudiov1.json>

The information is displayed in the projectUpdate() section.

Now Validate

At this point, the project should be setup and viewable using the Selene Network core code. To perform this validation, add “?contract=your_contract_address” to the URL of any install. For instance:

https://amorstyle.com/dsn/“?contract=your_contract_address”

The project should be displayed as expected.

If the project is not showing up as expected (mis-spelling or typos), correct the problem so the correct files are on the server and then adjust the SHA1 hash of that file in the project JSON file and then call updateProject() with the sample old path and file but a new SHA1 hash.

Building and Testing

In the case where you want to alter the sample code to add or alter the smart contract functionality, you’ll want to read over this section to get an idea regarding how to test the contract before launching.

Prerequisites

In order to make changes to the smart contract, the reader is expected to have a basic understanding of the Solidity programming language and using Remix for Theta.

The Environment of Remix

The user will want to use the Remix version offered through ThetaScan.io. Visit the “Smart Contract HQ” section and click on “Remix for Theta.”

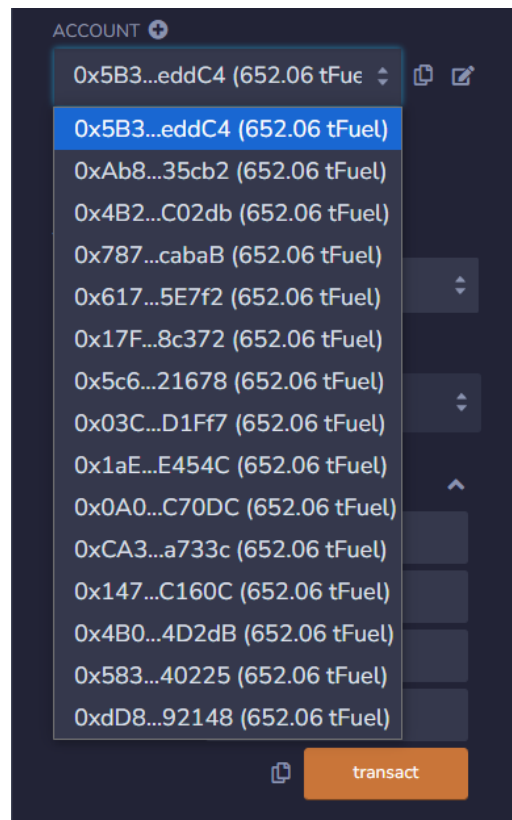
In the Theta Remix IDE: Create a workspace, copy all the sol files from the project in using file names that match the project. Then compile the sol file that has the project name. It is recommended that you use the Advance Configuration with the EVM Version set to ‘berlin’ and “enable optimization” of 200.

You’ll also want to compile, using the same settings, the file test_env1m.sol. This is a contract that will be used to fake the interactions that would be performed with the core Selene Network project files.

Note that both contracts should compile without errors or warnings.

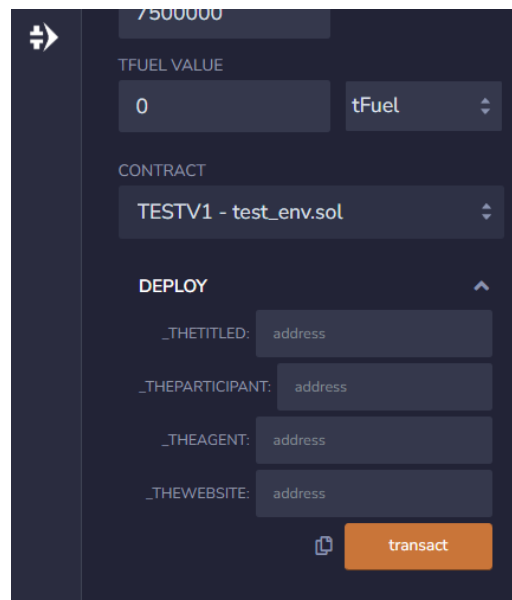
Deploying for Testing

The Remix development environment provides a number of accounts that can play roles that allow you to test the functionality of your change. If you click the dropdown listbox under ACCOUNT you’ll get something like the following.

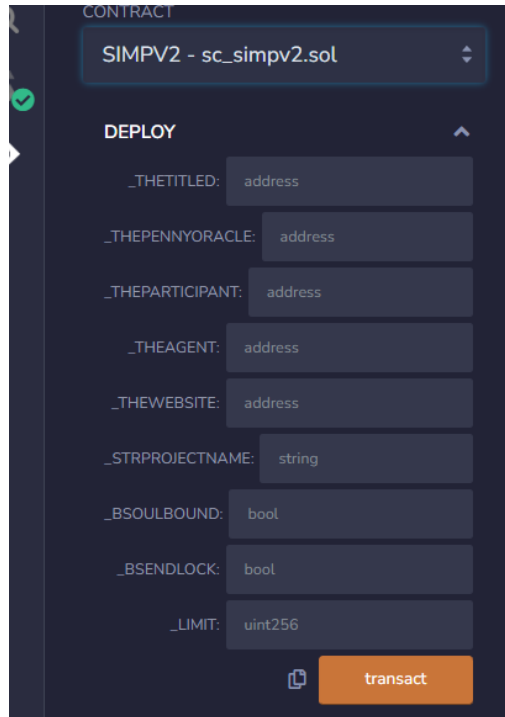


Take note of these accounts for you will want to pick out a titled, an owner (address that launches the contract), and customer.

When you deploy the test environment contract, you'll need to provide accounts from the above list to play different roles. You'll want to read over the test code, but when you put an account in these different fields, the test code will pretend that these accounts are of that type of contract.



After deploying the Test contract, you'll want to deploy the project itself. All of these fields are covered in this document above. The only difference between launching on chain and in the test environment of Remix is that you provide the contract of the test environment for `_thePennyOracle`, `_theParticipant`, `_theAgent` and `_theWebsite`. The test contract is going to pretend to be all these different projects for you.



CONTRACT

SIMPV2 - sc_simpv2.sol

DEPLOY

_THETITLED: address

_THEPENNYORACLE: address

_THEPARTICIPANT: address

_THEAGENT: address

_THEWEBSITE: address

_STRPROJECTNAME: string

_BSOULBOUND: bool

_BSENDLOCK: bool

_LIMIT: uint256

transact

After you deploy the contract, you'll need to perform both the `payInitialize()` and the `projectUpdate()` calls in order to setup the contract for testing.

When testing, it is useful to set the cost of minting to a small value. This way the wallets will usually have enough funds to play.

Ultimate Goal

After the testing has validated that the new code is working as intended, you'll want to make sure the project that you host matches the project that you just built.

Make sure you copy the content of each modified file back into your development folder into the file of the same name. If you add a new file, replicate that source back into your project development folder.

Gather both the ABI and bytecode and place it into the appropriate file in your development folder.

Note that after you launch your smart contract to the Theta blockchain the high-level Selene Network code will reference your ABI file in the UI. Make sure that your ABI is the project name plus “`_abi.json`” in the project directory.

Launching

Once you've saved your bytecode and ABI, visit ThetaScan's 'Smart Contract HQ' and 'Deploy Contract' using this new bytecode and ABI.

Then, configure it as needed referencing the information above as needed.

Disclaimer

As with any code, there could be bugs. Use this code at your own risk. If you find a bug, please fix it, and then let people know what the fix is. Also, this project depends upon other third-party tools and utilities, which could have or develop blocking bugs. Thus, use this code at your own risk.