RADIX

SCRYPTO DEVELOPER EVENT

Scrypto Challenge Guide

Table of Content

Introduction

This morning's workshop presented many of the essential Scrypto concepts you will need when writing your first Scrypto blueprint. In addition to that, **we demonstrated live a token sale blueprint being written in Scrypto**.

This blueprint showcases how tokens (known as resources in the Radix ecosystem) can be created through the `ResourceBuilder`, how these tokens can be sold in exchange for XRD, and how administrative and privileged access can be implemented in Scrypto such that only the original seller of the tokens is able to withdraw the funds they obtained from the sale.

In this portion of the event, we challenge you to build the same token sale demonstrated earlier with the same interface (the same methods and functions).

➔ **Use this document** as a guide of what you should build, the different methods and functions that should exist on your blueprint, the different state variables, and the specifications of the tokens to conduct the sale for.

- ➔ **Team up.** This is a guided team challenge meaning that you are encouraged to work in a team of up to 4 people. More team members = larger prizes to win!*
- ➔ **Need help?** **Join Discord** to ask questions. The radix team and community will be answering your questions right away! Best place to start is the #scrypto-rust-beginners channel!

**Please note that you can only be joining one team to complete the challenge. Your Eventbrite ticket number will be used as a unique identifier.*

Resource Specifications

This section defines the specifications of the resource (token) that you should create for the token sale.

Metadata

As part of this challenge, the resource that you create needs to have a number of required metadata fields such as the name of the participating team and the ticket numbers of all of the members in the team. The following are the metadata fields that your resource can have.

Table 1: A table showing the metadata fields that your token can have.

Name	Functionality
`name`	The name that you wish to give to your token. This should ideally by the name of your team.
`team-member-1-ticket-number`	The ticket number of the first team member. If you are working alone and not in a team, then this is your ticket number.
`team-member-2-ticket-number` (Optional)	An optional field of the ticket number of the second member of the team. If your team has less than two people, then this field should not be included.
`team-member-3-ticket-number` (Optional)	An optional field of the ticket number of the third member of the team. If your team has less than three people, then this field should not be included.
`team-member-4-ticket-number` (Optional)	An optional field of the ticket number of the fourth member of the team. If your team has less than four people, then this field should not be included.

Permissions and Authorization

The token that you create does not need to have any additional behavior beyond that which is provided by default.

Supply

The token should have a fixed supply of 100,000 tokens that are minted on the fly when the resource is first created.

Divisibility

The token should have a divisibility of 18 (`DIVISIBILITY_MAX`).

Blueprint Specifications

This section of the document provides you with the blueprint specifications and guides you through what you should build and leaves the implementation details up to you. In this challenge, you need to write a token sale blueprint that creates a token and then allows for this token to be sold in exchange for XRD.

When a component is first instantiated from this blueprint, the new token is created alongside any needed badges to prove ownership of the token sale. Some initial supply is defined for the token and this initial supply is stored in a vault in this component.

The component has a method called `buy` which allows users to purchase your token in exchange for XRD. When tokens are purchased, the paid XRD is stored in an XRD vault in the component and

Table 2: A table containing the specifications of the different methods and functions that you should implement on your blueprint.

Name	Type	Who can call it	Arguments	Returns	Functionality
`new`	Function	Anyone	1. `price_per_token`: This is a `Decimal` that defines the price of your tokens in XRD.	1. `ComponentAddress`: The address of the newly created `TokenSale` component. 2. `Bucket`: Contains the seller badge we would like to provide the seller to allow them to authenticate themself later on.	In object-oriented programming languages, there is a concept called *constructors* which are class functions that create new objects. The `new` function can be thought of as a *constructor* which creates new `TokenSale` components from the `TokenSale` blueprint and initializes the component with all of the required information and data. When this function is called, it first creates two resources: 1. **Seller Badge:** This is the badge that the seller can use to change the price of tokens and to withdraw the funds they've obtained from the sale. 2. **Your Token:** This is the token which the component will be doing a sale for. This token should be created according to the specifications defined in the "*Token Specifications*" section. After the tokens are created, this function defines the `AccessRules` of the token sale component that is being created such that the methods `chance_price` and `withdraw_funds` can only be called by a holder of the seller badge. With the `AccessRules` defined, the component is instantiated and globalized.
`buy`	Method	Anyone	1. `funds`: This is a `Bucket` of XRD which contain the amount of funds the buyer would like to spend to buy the tokens.	1. `Bucket:` Contains the tokens which have been bought with the provided XRD.	This method powers the `TokenSale` component and allows for the sale to take place. Buyers send this method a `Bucket` of XRD and this method returns back a `Bucket` of "Your Token". When the buyers send their `Bucket` of XRD, this method uses the `price_per_token` to determine how much of "Your Token" should be sent back in exchange for the amount of XRD sent. As an example, at a `price_per_token` of 0.5, when 10 XRD is sent, this method sends back 20 of your tokens. As such, the functionality of this method is quite simple:

					1. This method calculates how much of “Your Token” can be sent back in exchange for the amount of XRD sent. 2. It deposits the XRD into its XRD `Vault`. 3. It takes the calculated amount of “Your Token” from its `Vault` and returns it as a `Bucket`.
`change_price`	Method	Seller	1. `price`: This is a `Decimal` that defines the new price of tokens that the seller wishes for the tokens to be sold at.		This is an authenticated method that can only be called by the original seller of the tokens to update the price at which the tokens are being sold. This method is authenticated through the `AccessRules` set when the component is first instantiated through the `new` function.
`withdraw_funds`	Method	Seller	1. `amount`: This is a `Decimal` of the amount of XRD that the seller wishes to withdraw from the component.	1. `Bucket`: Contains XRD tokens of the requested amount.	This is an authenticated method that can only be called by the original seller of the tokens to withdraw the funds that they have obtained from the sale of the tokens. This method is authenticated through the `AccessRules` set when the component is first instantiated through the `new` function.

To aid you in this challenge, *listing 1* provides you with a “blueprint skeleton” which is a blueprint missing the state variables and the implementation. You can use this skeleton to build your blueprint and to verify the arguments and expected returns of the different methods and functions that this blueprint has.

```rust
use scrypto::prelude::*;

blueprint! {
   struct TokenSale {
       // Your state variables go here.
   }

   impl TokenSale {
       pub fn new(price_per_token: Decimal) -> (ComponentAddress, Bucket) {
           // Your `new` implementation goes here.
       }

       pub fn buy(&mut self, funds: Bucket) -> Bucket {
           // Your `buy` implementation goes here.
       }

       pub fn withdraw_funds(&mut self, amount: Decimal) -> Bucket {
           // Your `withdraw_funds` implementation goes here.
       }

       pub fn change_price(&mut self, price: Decimal) {
           // Your `change_price` implementation goes here.
       }
   }
}
```

Listing 1: A code listing of a “skeleton blueprint” which contains the signature of the different methods and functions on this blueprint.

Judging Criteria

A blueprint would be deemed to work if it satisfies all of the following requirements:

1. If the blueprint interface is implemented according to Table 2.
2. If the token is implemented according to the “resource specification” section.
3. If the blueprint compiles.
4. If the blueprint functions as expected.
    a. Tokens can be bought from the component at the price specified at instantiation.
    b. Only the seller is able to withdraw the funds accumulated by the component.
    c. Only the seller is able to change the price of the tokens.
    d. Changing the price of the tokens reflects on future sales.

How to Submit

Submission Deadline: Saturday 23rd, August 2022, 18h Munich (GMT+2). (17h Lisbon)

1. Install the Scrypto toolchain and make sure that you are on v0.4 (or later) with “scrypto –version”.
2. Fork the challenges repository.
3. Clone the forked repository into your local environment.
4. In your terminal, navigate to the “smaller-challenges” directory and create a new Scrypto package with your team name through “scrypto new-package [your-team-name]”.
5. When you are done with your code, commit it, and push it to GitHub. Then, open a pull request with your code.
6. Fill the form here with a link to your pull request.

Note: each team should submit a single pull request and should submit the form only once.

Note: you can only be joining one team to complete the challenge. If a member joins several teams, only the first submission will be accepted.

Note: each member needs to add their Eventbrite ticket number in the form as it will be used as a unique identifier. (ex.: #3917117809)

After you submit

1. Enjoy the after party!
2. We will review your team submission within a week.
3. If your submission is accepted, you will receive an email to “claim your token” with a form for you to fill.
4. We will email you again to let you know when your tokens have been transferred.

Resources

Setting up your Environment

If this is your first time building a dApp in Scrypto, then there are a few things that you will need to install to begin your journey with Scrypto:

1. Download and Install a compatible IDE such as Visual Studio Code + Rust Analyzer, or IntelliJ IDEA + Rust Extension.

2. Download and Install the Rust Toolchain.
3. Download and Install the Scrypto Toolchain.

Documentation and Examples

Regardless if you're new to Scrypto or you're a Scrypto veteran, you will find the Scrypto documentation to be a useful resource throughout your Scrypto journey.

Another resource that you will, no doubt, find helpful is the Scrypto examples in our official repository and our community repository. The examples in these repositories are pretty comprehensive and would be of help especially if you would like to get a feel for how asset-oriented applications can be developed in Scrypto.

Support

Do you have questions about the challenge or need help with Scrypto? Join Discord to ask questions. The radix team and community will be answering your questions right away! Best place to start is the **#scrypto-rust-beginners** channel!

Prizes

The prizes you get from completing this challenge vary depending on whether you are in a team or not. As your team grows, the reward is given to each team member for successful completion of the challenge grows as well. Therefore, you are encouraged to work in a team if you wish to maximize your prize. The following is the rewards table:

Table 3: The reward table of the amounts of reward given per team member for successful completion of the challenge.

Total Members in the Team	Reward per Member
1	$25 in XRD
2	$30 in XRD
3	$35 in XRD
4	$40 in XRD

Please note that you can only be joining one team to complete the challenge. Your Eventbrite ticket number will be used as a unique identifier.