0 Introduction

0.a Purpose of the Smart Contracts

0.a.1 Sponsorships

Sponsoring users is a way for BrightID to have a continuous stream of funding while allowing BrightID to remain a public good. It spreads the burden of funding BrightID to many participants.

To use BrightID's verification system, a user must be sponsored--which happens just once per user per lifetime. In the most common scenario, an application purchases a sponsorship on behalf of a user. Purchasing sponsorships is recorded on Ethereum, while assigning a sponsorship to a BrightID user is recorded on BrightID nodes.

0.a.2 Sponsorship Contexts

Every sponsorship has at most one context, which indicates which application controls it.

When a sponsorship is newly created, it has no context assigned to it. The holder of unassigned sponsorships can call a function on the smart contract to assign a context to some number of their sponsorships. To prevent abuse, a context can only be assigned to a sponsorship once (code).

0.a.3 Subscriptions

In addition to direct sales of sponsorships, a limited number of *subscriptions* will be available to early supporters. A subscription produces new sponsorships every month for nearly six years.

0.b Auditing Priorities

By far the most important contract to consider is the Sponsorships contract. It is designed to provide income for a long time. By contrast, the sale of subscriptions has a cap (code) and once subscriptions are activated they have a fixed lifetime. The Subscriptions contract is easier to fix if needed since the balances shouldn't change much--especially after the cap is reached.

The minter contracts (SponsorshipsMinter and SubscriptionsMinter) can be replaced using the functions of the MinterRole if issues are found after they have been deployed.

See also Handling Upgrades.

1 Sponsorships

1.a Name and Symbol

We will use the name **Sponsorships** to describe the asset (code). The symbol will be **Sp** (code).

1.b Transferability

Only sponsorships that haven't been assigned a context are transferable (code).

1.c Balances

The contract can return the number of Sponsorships assigned to a given context for a given address (code). The contract can return the entire balance of a context across all addresses (code).

A user's unassigned balance (code) should be visible from MetaMask (code, code, code, code).

All balances are visible from a web application.

1.d Purchase

Purchases are handled using the SponsorshipsMinter contract (code).

Sponsorships are purchased for 1 DAI each (code, deployment script constants). Tokens used to purchase sponsorships are immediately sent to BMAIN DAO (code, code, code, code, code, deployment script constants).

Newly purchased sponsorships are not assigned a context (code).

1.d.1 Refunds

The number of sponsorships purchased is rounded down to the nearest unit and any payment left over is refunded (code).

Where possible, other tokens and ether sent to the smart contract are reclaimable.

1.d.2 Changing Purchase Token and Price

The contract owner of the SponsorshipsMinter can change the purchase token (code) and price (code).

1.e Contexts

An address that holds sponsorships can have some that are assigned a context (code), and some that are unassigned.

An address can assign some number of their sponsorships to a given context (code).

2 Subscriptions

2.a Name and Symbol

We will use the name **subscriptions** to describe the asset (code). The symbol will be **Subs** (code).

2.b Activation

An address holder may activate any number of their subscriptions (code). After activation, subscriptions start generating sponsorships according to the schedule shown in "Claiming Sponsorships."

2.c Transferability

Only subscriptions that haven't been activated are transferable (code).

2.d Balances

The balance of unactivated subscriptions should be visible from MetaMask (code, code, code, code, code). An event is emitted when subscriptions are activated (code). This allows a web app to track how many subscriptions a user has activated.

2.e Purchase

Purchases are handled using the SubscriptionsMinter contract (code). Newly purchased subscriptions aren't activated (code).

2.e.1 Price Steps

The price increases in steps according to the following schedule. (code, code)

Step	Subscriptions	Price
1	400,000	1 DAI
2	500,000	2 DAI

2.e.2 Purchases spanning two steps

If a purchase would span two steps, only the subscriptions from the earlier step are purchased and the rest of the payment is refunded (code).

2.e.3 Refunds

The number of subscriptions purchased is rounded down to the nearest unit and any payment left over is refunded (code).

If a purchase is attempted after the sale has ended, the payment is refunded (code).

Where possible, other tokens and ether sent to the smart contract are reclaimable.

2.e.4 Direction of funds

Tokens used to purchase subscriptions are immediately sent to BMAIN DAO (code, code, code, code, code, deployment script constants)..

2.e.5 Batches

Subscriptions are activated in batches (code). The starting timestamp[1] for each batch must be recorded separately (code, code) and will be needed when sponsorships are claimed (code).

2.e.6 Claiming Sponsorships

New sponsorships are available from an activated subscription every month (starting with month zero--one sponsorship is claimable immediately), following this pattern:

Month (inclusive)	Sponsorships per month
0-11	1

[1] The timestamp doesn't have to be very precise. It could be months since October 1, 2019.

12-23	2
24-35	3
36-47	4
48-59	5
60-71	6

Months are actually thirty day periods. The final sponsorships can be claimed after 71 * 30 = 2130 days. One subscription produces 252 sponsorships in total (code).

The SubscriptionsMinter claim function calls the corresponding claim function on the Subscriptions contract which returns the number of sponsorships which should be added to the balance for the calling address (code, code, code).

The SubscriptionsMinter contract then calls the Sponsorships contract's mint function which increments the account's balance with new sponsorships accordingly (code, code).

When an address claims sponsorships, it claims all available sponsorships from all eligible batches (code).

3 Sending Tokens to BMAIN DAO

The Sponsorships (code), Subscriptions (code), SponsorshipsMinter (code), and SubscriptionsMinter (code) contracts all inherit from FinanceManager which allows tokens to be sent to a finance app using Aragon's deposit function definition (code, code, code). When used with an Aragon DAO, it records the transaction and deposits tokens in the vault. Transactions can have a reference telling what the transaction is for. Underlying _deposit function.

The SponsorshipsMinter and SubscriptionsMinter deposit tokens used for purchase, while all four contracts inheriting from FinanceManager allow any ERC20 tokens sent by mistake to be deposited.

The address of the finance app used by a contract can be changed by the owner (code).

4 Minters

The Sponsorships (code) and Subscriptions (code) contracts inherit OpenZeppelin's MinterRole. This means that the contract creator is designated as a Minter and Minters can add and remove

other Minters. The minter role will be transferred to BMAIN DAO's Agent App so that it acquires the ability to replace minters.

4.a Minters for the Sponsorships contract

The SponsorshipsMinter, SubscriptionsMinter, and BMAIN DAO's Agent App are Minters for the Sponsorships contract.

The SponsorshipsMinter needs to have the MinterRole to call the mint (code) function on the Sponsorships contract when someone wants to purchase sponsorships (code).

The SubscriptionsMinter needs to have the MinterRole to call the mint (code) function on the Sponsorships contract when someone wants to claim sponsorships from their subscriptions (code).

BMAIN DAO's Agent App needs to have the MinterRole to replace the SponsorshipsMinter or SubscriptionsMinter contracts if needed. This also means that BMAIN DAO's Agent App can mint sponsorships on behalf of any address (code).

4.b Minters for the Subscriptions contract

The SubscriptionsMinter and BMAIN DAO's Agent App are Minters for the Subscriptions contract.

The SubscriptionsMinter needs to have the minter role to call the mint (code) or claim (code) functions on the Subscriptions contract when someone wants to purchase Subscriptions (code) or claim Sponsorships (code), respectively.

BMAIN DAO's Agent App needs to have the MinterRole to replace the SubscriptionsMinter contract if needed. This also means that BMAIN DAO's Agent App can mint subscriptions on behalf of any address (code). It can also mark sponsorships from subscriptions as claimed (without actually minting them) for any address by calling Subscriptions.claim() directly.

4.c Replacing Minters

SponsorshipsMinter and SubscriptionsMinter can be replaced if issues are found after they have been deployed.

4.c.1 Detaching the SponsorshipsMinter contract

From the SponsorshipsMinter contract, disable purchases (code).

4.c.2 Detaching the SubscriptionsMinter contract

From the SubscriptionsMinter contract, disable purchases (code) and claims (code).

4.c.3 Adding New Minters

A new SponsorshipsMinter or SubscriptionsMinter can be added following the example of the initial deployment.

5 Pausing Certain Functions

Pausing purchases, transfers, claims and context assignments can be useful in order to freeze balances so contracts can be replaced. See replacing minters and handling upgrades.

5.a Pauser Role

Because they inherit from OpenZeppelin's ERC20Pausable, the creator of the Sponsorship (code) and Subscriptions (code) contracts acquires the Pauser role for those contracts (code) and can call the functions of the OpenZeppelin's Pausable module on them. The pauser role will be transferred to BMAIN DAO's agent app.

5.b Pausing Functions in the Sponsorships contract

If the Sponsorships contract is paused, the mint (code) and assignContext (code) functions will be disabled (code).

5.c Pausing Functions in the Subscriptions contract

If the Subscriptions contract is paused, the mint (code), claim (code), and activate (code) functions will be disabled (code).

6 Tokens Sent to Contracts by Mistake

The Sponsorships (code), Subscriptions (code), SponsorshipsMinter (code) and SubscriptionsMinter (code) contracts all inherit from FinanceManager which allows their owner to transfer ERC20 tokens sent to them by mistake to the BMAIN DAO (code).

7 Contract Ownership

Because the Sponsorships, Subscriptions, SponsorshipsMinter and SubscriptionsMinter contracts all inherit from FinanceManager, they also all inherit OpenZeppelin's Ownable (code). The eventual owner will be the BMAIN DAO's Agent App.

7.a Functionality provided only to the Owner

- Sending tokens sent by mistake to the BMAIN DAO (code).
- Setting the token and price used for purchasing Sponsorships (code, code).

8 Initial Deployment

8.a Deployment Script

The script to deploy the contracts will be executed by a BrightID team member. All special capabilities derived from being the creator of the contracts will then be transferred to BMAIN DAO's Agent app, leaving the team member who executed the script with no capabilities.

8.a.1 Constants

- The finance address for Sponsorships, Subscriptions, SponsorshipsMinter and SubscriptionsMinter contracts are set to BMAIN DAO's finance address (code, code).
- The initial token used for purchasing Sponsorships (code) and Subscriptions (code) is DAI (code).

8.a.2 Minter Roles

The SponsorshipsMinter contract is added as a Minter for sponsorships (code). The SubscriptionsMinter contract is added as a Minter for both Subscriptions (code) and Sponsorships (code).

8.b Transferring Capabilities to BMAIN DAO's Agent App

8.b.1 Owner Roles

The owner of the contracts has the capabilities listed here. The team member (initial owner) will need to call transferOwnership() on the Sponsorships, Subscriptions, SponsorshipsMinter, and SubscriptionsMinter contracts and supply the BMAIN DAO Agent App address.

8.b.2 Minter Roles

The team member (initial owner) should call addMinter() on both the Sponsorships and Subscriptions contracts and supply the BMAIN DAO Agent App address.

Next, the team member should call renounceMinter() on both the Sponsorships and Subscriptions contracts.

8.b.3 Pauser Roles

The team member (initial owner) should call addPauser() on both the Sponsorships and Subscriptions contracts and supply the BMAIN DAO Agent App address.

Next, the team member should call renouncePauser() on both the Sponsorships and Subscriptions contracts.

9 Auditing the Deployment

9.a Contract Deployment

9.a.1 Sponsorships Contract

Verify that code on etherscan matches the code in github. Verify that read and write contracts match this spec.

9.a.2 Subscriptions Contract

Verify that code on etherscan matches the code in github. Verify that read and write contracts match this spec.

9.a.3 SponsorshipsMinter Contract

Verify that code on etherscan matches the code in github. Verify that read and write contracts match this spec.

9.a.4 SubscriptionsMinter Contract

Verify that code on etherscan matches the code in github. Verify that read and write contracts match this spec.

9.b Roles and Finance Managers

9.b.1 Sponsorships Contract

Verify that the events on etherscan result in BrightID Main DAO's agent app (0x005ca4f1493f451c1588186c570a434ec38e718d) being an owner, pauser, and minter. The SponsorshipsMinter contract (0xF6942Db401abFbaBA0252c15c25FC3729D1A467b) and SubscriptionsMinter contract (0x200C142d18892cE0589e2bD93617A524692AC5E3) should also be minters. Verify that any other pausers or minters have been renounced. Verify that the finance manager is set to BrightID Main DAO's finance app (0x8b2bc1aa673aae1ec9c75704f9ad2a475804cec8).

9.b.2 Subscriptions Contract

Verify that the events on etherscan result in BrightID Main DAO's agent app (0x005ca4f1493f451c1588186c570a434ec38e718d) being an owner, pauser, and minter. The SubscriptionsMinter contract (0x200C142d18892cE0589e2bD93617A524692AC5E3) should also be a minter. Verify that any other pausers or minters have been renounced. Verify that the finance manager is set to BrightID Main DAO's finance app (0x8b2bc1aa673aae1ec9c75704f9ad2a475804cec8).

9.b.3 SponsorshipsMinter Contract

Verify that the events on etherscan result in BrightID Main DAO's agent app (0x005ca4f1493f451c1588186c570a434ec38e718d) being the owner. Verify that the finance manager is set to BrightID Main DAO's finance app (0x8b2bc1aa673aae1ec9c75704f9ad2a475804cec8).

9.b.4 SubscriptionsMinter Contract

Verify that the events on etherscan result in BrightID Main DAO's agent app (0x005ca4f1493f451c1588186c570a434ec38e718d) being the owner. Verify that the finance manager is set to BrightID Main DAO's finance app (0x8b2bc1aa673aae1ec9c75704f9ad2a475804cec8).

10 Other Considerations

10.a Handling Upgrades

10.a.1 Sponsorships or Subscriptions Contracts

After pausing functions in the Sponsorships contract or the Subscriptions contract (as the case may be), a contract similar to OpenZeppelin's ERC20 Migrator could be used to migrate balances before deploying a new contract

10.a.2. SponsorshipsMinter or Subscriptions Minter Contracts

These contracts can be replaced using the instructions in the "Replacing Minters" section.

10.b. Pros and Cons of Transferability

Despite the arguments below, any non-transferable token can be transferred if its holder is a smart contract--or the address itself could be transferred. Limiting transferability might be done as a token attempt to comply with a law, and providing transferability might be done to make what was always possible more convenient.

It's useless (and inconvenient) to try to make a token non-transferable. Only courts can punish someone for transferring something that should be non-transferable.

10.b.1 Sponsorships Transferability

Pros to Transferability	Pros to Non-Transferability
Easy to resell, gift, donate	Early on (before there is a market to pay someone else to assign a context), if there is an exploit, an attacker will have less opportunity to sell their sponsorships before we can fix the exploit.
Businesses might be inclined to buy more Sponsorships if they can easily resell the ones they don't use.	Potentially less undercutting of the smart contract price for sponsorships (by those who got cheaper sponsorships from subscriptions) if it's harder to sell them.
A user can transfer to another wallet they control for convenience.	

If transfers are not allowed, there will probably be a market for paying someone else to assign a context to their sponsorships to what the buyer chooses. Allowing transfers makes it easier to accomplish this by simply trading the sponsorships.	
Can use the existing ERC20 specification without modification. (Non-transferable tokens are not ERC20 compliant because they omit certain functions.)	

10.b.2 Subscriptions Transferability

Pros to Transferability	Pros to Non-Transferability
Easy to resell, gift, donate	If there is an exploit, an attacker will not be able to sell the subscriptions, but only get the sponsorships they can claim before we can fix the exploit.
Could create more excitement to buy in the first price step in hope of a quick flip once the second price step is reached.	The price of the second step won't be undercut by people who bought in the first step trying to flip.
Allows for secondary markets for unactivated subscriptions.	Easy to match up buyers of subscriptions to claimers of sponsorships (they will be the same) for tracking deferred income.
Can use the existing ERC20 specification without modification. (Non-transferable tokens are not ERC20 compliant because they omit certain functions.)	Transferability is required in some countries for an asset to be considered a "security" (and therefore require a prospectus). Being non-transferable rules that out.
A user can transfer to another wallet they control for convenience.	